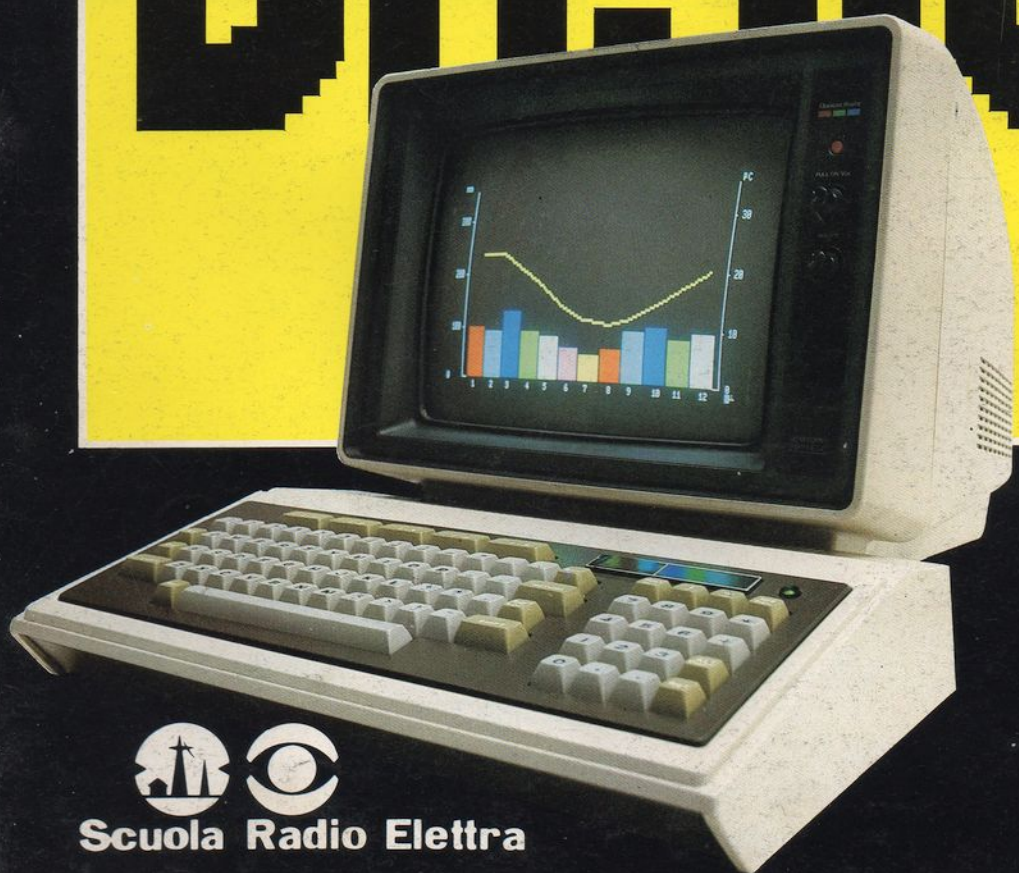
# PARLA BASIC



Scuola Radio Elettra

Scuola Radio Elettra
Torino

# Analisi e programmazione

Lezione 3

1 — La programmazione elettronica sul computer
2 — Che cos'è la programmazione
3 — Analisi e programmazione
4 — Il diagramma di flusso
5 — Nozioni sull'organizzazione della memoria del microcomputer
6 — Esempi di programmazione
7 — Linguaggi di programmazione e sistemi operativi
8 — Generalità e struttura del BASIC

## Che cosa apprenderà

In questa lezione saranno presi in considerazione i concetti fondamentali della programmazione del computer.
Mediante opportuni esempi, avrà cosí modo di apprendere, di passo in passo, i primi elementi della programmazione.
A questo scopo verrà dettagliatamente esaminata la costituzione del diagramma di flusso (flow-chart), che raffigura schematicamente la serie completa delle istruzioni (algoritmo) necessarie per risolvere un dato problema con il computer.
La lezione proseguirà con la descrizione dei linguaggi di programmazione piú diffusi: ASSEMBLY, FORTRAN, COBOL, APL, ecc.
Completerà la lezione una sintesi di tutti i caratteri piú importanti delle istruzioni del linguaggio BASIC.

## 1. – La programmazione elettronica sul computer

Si è piú volte ribadito in precedenza che il calcolatore, anche se lavora in maniera automatica, è una **macchina programmata.**

Ma da chi è programmata e come è programmata? Può sembrare un giro banale di parole: il calcolatore è programmato da un programma, "programmato" a sua volta da un programmatore.

Il programma "programmato" è un insieme ordinato di istruzioni, scritte in un frasario simbolico (linguaggio simbolico) in base a regole sintattiche che vengono preparate e formulate in una fase precedente all'elaborazione automatica.

Normalmente, il programma si presenta come un documento costituito da uno o piú moduli, su cui vengono compilate le istruzioni di programma usando le regole di un particolare linguaggio simbolico; ci sono di conseguenza moduli per il linguaggio FORTRAN, per il COBOL, per l'ASSEMBLY, per il BASIC, ecc.

Nella *fig. 1* è riportata, a titolo esemplificativo, l'impaginazione di un programma scritto in COBOL.

Come si può rilevare, all'inizio della pagina sono riportate indicazioni di carattere generale, come nome e cognome dell'autore (cioè del programmatore) e denominazione del programma.

PROCEDURA
PROGRAMMA
PROGRAMMATORE

FOGLIO DI PROGRAMMAZIONE
COBOL.

IDENTIFICAZIONE PROGRAMMA

SEQUENZA PAGINA RIGA

```
IDENTIFICATION DIVISION.
PROGRAM-ID. LISTA1
AUTHOR.
    GONNELLI GIANCARLO.
REMARKS.
    IL PROGRAMMA EFFETTUA UNA LISTA DEI CLIENTI.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT ANACLI ASSIGN SYS010-UT-3340-S.
    SELECT STAMPA ASSIGN SYS011-UR-1403-S.
DATA DIVISION.
FILE SECTION.
FD  ANACLI
        RECORDING F
        BLOCK CONTAINS 12 RECORDS
        LABEL RECORDS STANDARD
        DATA RECORD REC-CLI.
01  REC-CLI.
    02 CODCLI                    PIC 9(6).
    02 NOMECLI                   PIC X(25).
    02 INDIRCLI                  PIC X(25).
    02 CAPCLI                    PIC 9(5)     COMP-3.
    02 LOCALCLI                  PIC X(15).
FD  STAMPA
        RECORDING F
        LABEL RECORD OMITTED
        DATA RECORD RECSTAM.
01  RECSTAM.                     PIC X(132)
WORKING-STORAGE SECTION.
01  CLICOMO.
    02 CODCOMO                   PIC 9(6).
    02 NOMECOMO                  PIC X(25).
    02 INDIRCOMO                 PIC X(25).
    02 CAPCOMO                   PIC 9(5)     COMP-3.
    02 LOCALCOMO                 PIC X(15).
01  RIGAST.
    02 CODST                     PIC 9(6).
    02 FILLER                    PIC XXXX     VALUE SPACE.
    02 NOMEST                    PIC X(25).
    02 FILLER                    PIC XXX      VALUE SPACE.
    02 INDIRST                   PIC X(25).
    02 FILLER                    PIC XXX      VALUE SPACE.
    02 CAPST                     PIC 9(5).
    02 FILLER                    PIC XXX      VALUE SPACE.
    02 LOCALST                   PIC X(15).
    02 FILLER                    PIC X(33)    VALUE SPACE.
PROCEDURE DIVISION.
INIZIO.
    OPEN INPUT ANACLI
         OUTPUT STAMPA.
LEGGI.
    READ ANACLI INTO CLICOMO
         AT END GO TO CHIUDI.
    MOVE CODCOMO TO CODST.
    MOVE NOMECOMO TO NOMEST.
    MOVE INDIRCOMO TO INDIRST.
    MOVE CAPCOMO TO CAPST.
    MOVE LOCALCOMO TO LOCALST.
    MOVE SPACE TO SPAZIO.
    WRITE RECSTAM FROM RIGAST
         AFTER POSITIONING SPAZIO.
    GO TO LEGGI.
CHIUDI.
    CLOSE ANACLI STAMPA.
          STOP RUN.
```

*Fig. 1 – Aspetto del primo foglio di programmazione di un programma in linguaggio* COBOL.

Il programma potrebbe far parte di un complesso piú vasto di programmi, detto **procedura,** e quindi, sempre in testa al foglio di programmazione, si potrebbero indicare vari riferimenti per collegare il singolo programma alla procedura stessa.

Nella parte sottostante all'intestazione sono elencate le istruzioni del programma compilato nel linguaggio simbolico prescelto; ogni simbolo, carattere, numero di un'istruzione va scritto chiaramente nell'apposita casella del foglio di programmazione, cosí come è richiesto dalla sintassi del linguaggio simbolico.

Nel programma vengono riportati non solo i dati da elaborare, ma anche le istruzioni operative, compresa quella di "non rispettare" il programma, ossia di saltare da un'istruzione all'altra quando le particolarità del calcolo lo richiedono.

**La perforazione del programma**

Una volta compilato, il programma passa normalmente in PERFORAZIONE, cioè viene codificato su un altro supporto: la SCHEDA PERFORATA, costituita da un cartoncino rettangolare di circa 183×81 mm, su cui il bit "1" è rappresentato con un foro ed il bit "0" dall'assenza del foro.

Il programma diventa cosí un pacco di schede perforate, che può essere successivamente introdotto nella memoria del computer attraverso l'organo di lettura: il FOTOLETTORE.

Questa speciale apparecchiatura traduce il codice costituito dalle perforazioni in corrispondenti impulsi elettrici da inviare all'unità centrale di elaborazione del calcolatore. Il funzionamento del fotolettore è schematizzato nella *fig. 2.* La scheda intercetta la luce proiettata da un'apposita lampada, fatta esclusione per i fori attraverso i quali vengono eccitate le fotocellule corrispondenti che generano il segnale elettrico.

Se il sistema d'elaborazione non è dotato di lettore di schede, il programma può essere immesso gradualmente nell'unità centrale tramite altri organi di input, quali tastiere, telescriventi, ecc.: l'elaborazione di questo tipo è denominata **time-sharing** (ad intervalli di tempo), mentre quella che comporta l'immissione di dati e programmi attraverso schede è chiamata **elaborazione batch.**

Il programma codificato dagli organi d'ingresso passa nella memoria principale e subisce un'ulteriore codifica da parte di altri programmi, detti TRADUTTORI. In questa fase il programma scritto in linguaggio simbolico, ossia in un linguaggio simile il piú possibile a quello naturale

usato dall'uomo, **viene tradotto in linguaggio macchina,** cioè in un sistema di rappresentazione binaria, formato dai simboli 1 e 0, il solo che può essere elaborato dal computer.

**Inizio dell'elaborazione**

L'ordine di elaborazione viene impartito in tanti modi: il sistema piú comune consiste nel battere su un organo di input un'istruzione particolare, che ha il significato di **lancio del programma** (come per esempio l'istruzione **RUN**), oppure nel premere un tasto particolare della tastiera.

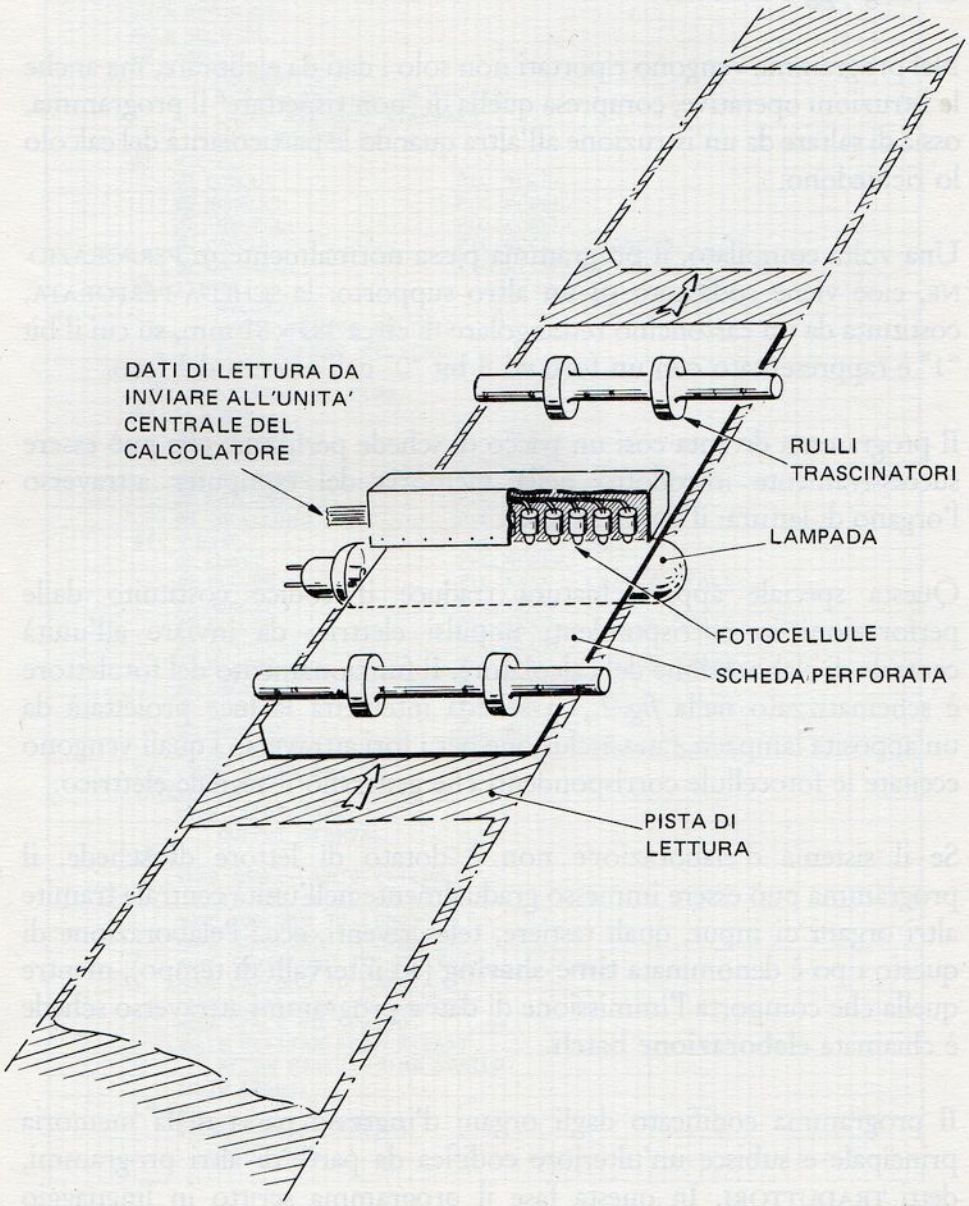



*Fig. 2 – Rappresentazione schematica del funzionamento del fotolettore di schede perforate.*

Se il programma che si vuole eseguire fa parte di una procedura d'elaborazione piú vasta, esso può essere eseguito da qualche istruzione "esecutiva", contenuta nella procedura stessa.

**L'elaborazione**

Impartito il comando di inizio, le istruzioni del programma, dislocate nella memoria in successione a partire da una locazione iniziale (indirizzo iniziale del programma), vengono prelevate una alla volta ed eseguite dall'unità di governo del computer.

Questa unità è il vero e proprio motore del calcolatore, perché controlla il flusso logico dell'elaborazione, mentre la memoria funge da deposito e l'unità di calcolo logico-aritmetica (ALU) esegue un numero limitato di operazioni binarie: addizione, sottrazione, moltiplicazione, confronti, trasferimenti.

## 2. - Che cos'è la programmazione

Il concetto di programmazione assume un significato tutto particolare se si riferisce al calcolatore elettronico.

Infatti, quando in genere si parla di programmazione, senza specificare né l'oggetto né i mezzi che si usano per programmare, si pensa normalmente ad un complesso di operazioni e di procedure svolte in un certo ordine sequenziale ed atte a realizzare determinati obiettivi futuri.

La programmazione economica in campo aziendale, la formulazione di programmi di sviluppo da parte di autorità pubbliche, i piani che normalmente ognuno fa durante la propria esistenza per raggiungere determinati obiettivi, sono tutti esempi di programmazione (qualcuno potrebbe definirli "programmi ad esito incerto").

Con questi programmi ci si limita a definire obiettivi e strumenti di massima, con la "speranza" piú o meno fondata di raggiungere almeno in parte le mete prefissate.

La programmazione sul computer è invece un concetto diverso. In questo caso **uno strumento preciso viene programmato perché fornisca i risultati attesi.** Volendo fare un esempio molto banale, se si programma il computer per eseguire la somma 2+2, si deve essere certi che il risultato fornito sia proprio 4. A questo scopo lo strumento (cioè il

computer) deve poter interpretare il programma in un modo **inequivocabile** per riuscire a fornire risultati altrettanto certi, non opinabili.

Come si fa a programmare questo strumento in modo non equivocabile? Semplicemente fornendogli regole precise (istruzioni) che abbiano le caratteristiche di seguito descritte.

**Caratteristiche delle istruzioni su computer**

a) **Le istruzioni devono essere scritte con un frasario di tipo algebrico;** quelle elaborative, infatti, sono scritte come delle relazioni matematiche: per esempio, per assegnare alla variabile A il valore proveniente dal prodotto della costante 3 per un'altra variabile Y, si scrive A=3*Y.

Anche le istruzioni di tipo logico od imperativo (che servono a variare il flusso dell'elaborazione) sono scritte in maniera inequivocabile: si calcola il valore di una certa relazione algebrica, per esempio di un'equazione o disequazione; se questo valore è 0 (zero), il programma prosegue per un certo punto, se è diverso da 0 prosegue per un altro punto.

**La sintassi (o grammatica) di un linguaggio di programmazione**

b) **Le istruzioni devono essere scritte secondo una sintassi precisa, che non consenta interpretazioni diverse da parte del calcolatore.** Per sintassi di un linguaggio di programmazione si intende il complesso delle regole formali in base alle quali vengono scritte le istruzioni.

L'imprecisione sintattica in cui noi spesso cadiamo quando scriviamo un'espressione algebrica non viene "perdonata" dal calcolatore.

Per esempio, se scriviamo un'istruzione di assegnazione nel seguente modo: Y=3X+1, il calcolatore segnala ERRORE DI SINTASSI, perché nel linguaggio di programmazione il prodotto della costante 3 per la variabile X si scrive:

3*X.

**L'algoritmo**

c) **L'algoritmo di risoluzione** (cioè la parte del programma che risolve il problema proposto) **deve convergere alla soluzione del problema in un numero finito di passi.** Se questa condizione non è soddisfatta, l'algoritmo usato non è adatto a risolvere il problema o non esiste soluzione al problema.

L'**algoritmo** è costituito da una successione finita di istruzioni,

formulate in modo chiaro ed esatto, che, eseguite dal calcolatore nell'ordine prestabilito, consentono di ottenere il risultato del problema proposto.

Non basta però fornire regole precise al calcolatore (cioè scrivere correttamente il programma nel linguaggio simbolico stabilito), ma è assolutamente necessario aver effettuato a monte un'analisi accurata del problema che si vuole elaborare; a questo scopo esistono strumenti d'analisi appropriati, come ad esempio il **flow-chart** (diagramma di flusso del programma).

**Il flow-chart**

Il flow-chart (si pronuncia approssimativamente flóu-ciárt) è una raffigurazione schematica del flusso di elaborazione; con esso è possibile controllare, verificare, correggere i passi di programma con una immediatezza straordinaria, che non sarebbe consentita rileggendo le istruzioni simboliche del programma stesso.

Il flow-chart rappresenta praticamente l'espressione visiva di un algoritmo, cioè la serie completa delle istruzioni necessarie a risolvere con il computer, passo per passo, un dato problema.

Il flow-chart, che sarà descritto piú dettagliatamente nel seguito della lezione, costituisce la "mappa" del programma.

La prima cosa che il programmatore deve quindi fare, prima ancora di iniziare a scrivere le istruzioni simboliche, è **studiare a fondo il problema da elaborare,** cioè capire bene la natura o essenza del problema. E' questo un passo veramente fondamentale per riuscire a programmare il computer.

Per fare ciò occorre, in primo luogo, tenere presente i seguenti due aspetti.

**Natura digitale del problema da risolvere**

a) **Il problema deve essere di natura digitale o riconducibile ad essa, cioè deve consistere in operazioni logico-matematiche su caratteri numerici ed alfabetici.** Un problema sentimentale, spirituale, giuridico, tanto per fare qualche esempio, non può essere trattato sul computer; un problema di salute può essere risolto soltanto se lo si trasforma in un problema probabilistico: in tal caso, si introducono dati statistici inerenti alla malattia e si ottengono, con appositi programmi statistici, risposte probabilistiche, cioè con certi margini d'errore.

**Utilizzazione generalizzata del programma**

b) **Il programma realizzato deve risolvere non soltanto un unico problema, ma una classe di problemi.** Non ha senso infatti usare il computer per risolvere problemi specifici senza tenere conto di possibili generalizzazioni. Se per esempio si programma il computer per trovare le radici di una certa equazione di secondo grado, il programma dovrà essere valido per qualsiasi altra equazione di secondo grado. Il programmatore, nella stesura del programma, dovrà quindi prevedere un input di parametri generale il piú possibile.

Dopo di ciò, si procederà alla fase successiva, cioè alla stesura del diagramma di flusso del problema. Da questo primo documento si vede già se il programma, prima ancora di essere elaborato dal computer, funziona.

Di seguito è riportato un semplice esempio di programmazione; di esso ci si è limitati a tracciare il diagramma di flusso del problema, senza scrivere il linguaggio simbolico delle istruzioni. Questa fase si chiama anche **codificazione,** cioè traduzione del flusso di programma riportato nel flow-chart in codici simbolici.

**Calcolo della lunghezza di una circonferenza**

Il semplice problema proposto consiste nel trovare la lunghezza della circonferenza di un cerchio, data la lunghezza del raggio.

Si sa dalla geometria che la formula risolutiva per trovare la lunghezza della circonferenza è la seguente:

$$C = 2 \pi r$$

dove:

C = lunghezza della circonferenza (incognita);
2 = costante;
$\pi$ = 3,14159 (costante);
r = raggio della circonferenza (dato d'ingresso).

Il programma da realizzare deve essere di tipo generale, cioè utilizzabile per tutta la classe di problemi che richiedono il calcolo della circonferenza quando sia noto il raggio; di conseguenza, esso deve prevedere l'introduzione di un parametro variabile da un organo di input, per esempio la tastiera; successivamente elaborerà la formula riportata sopra ($C = 2 \pi r$); infine stamperà il risultato (C) sulla stampante o sul monitor.

Per la programmazione del computer vengono usati due diagrammi base: il **diagramma a blocchi** del problema da risolvere, corredato di specifiche descrizioni (*fig. 3*), ed il **flow-chart**, cioè il diagramma di flusso delle elaborazioni, dotato anch'esso di ulteriori chiarimenti (*fig. 4*).

Le funzioni svolte dai tre blocchi che costituiscono il diagramma della *fig. 3* sono le seguenti.

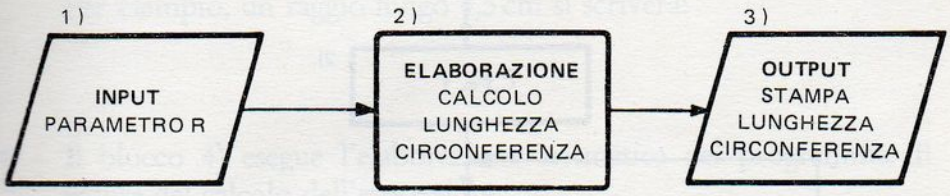



*Fig. 3 – Diagramma a blocchi.*

1) INPUT: introduzione di R, raggio di un cerchio.

2) ELABORAZIONE: calcolo della lunghezza della circonferenza di raggio R.

3) OUTPUT: stampa del valore della lunghezza della circonferenza appena calcolata.

Per quanto concerne il diagramma di flusso, osservando la *fig. 4* si può rilevare che esso è costituito da un insieme di figure geometriche diverse (blocchi), unite da linee ad un estremo delle quali è riportata una freccia; tali linee indicano le interconnessioni fra la successione delle istruzioni, ciascuna delle quali è appunto rappresentata con una figura geometrica nel cui interno è descritta l'operazione svolta.

Si analizzeranno ora le operazioni simbolizzate da ciascuna figura geometrica, o blocco, costituente il diagramma della *fig. 4*.

**L'input**

1° – Nel blocco 1) del diagramma di flusso del programma compare la scritta: INPUT:N. Ciò significa che si deve introdurre con un organo di input (per esempio con la tastiera) un numero chiamato N, il quale indica quanti problemi si devono risolvere, cioè quante lunghezze di circonferenza il programma deve calcolare.

2° – Il blocco 2) serve ad inizializzare il programma di elaborazione. Siccome sono previste N elaborazioni, occorre che un contatore indice (nel nostro caso l'indice di nome I) assuma prima il valore 1,

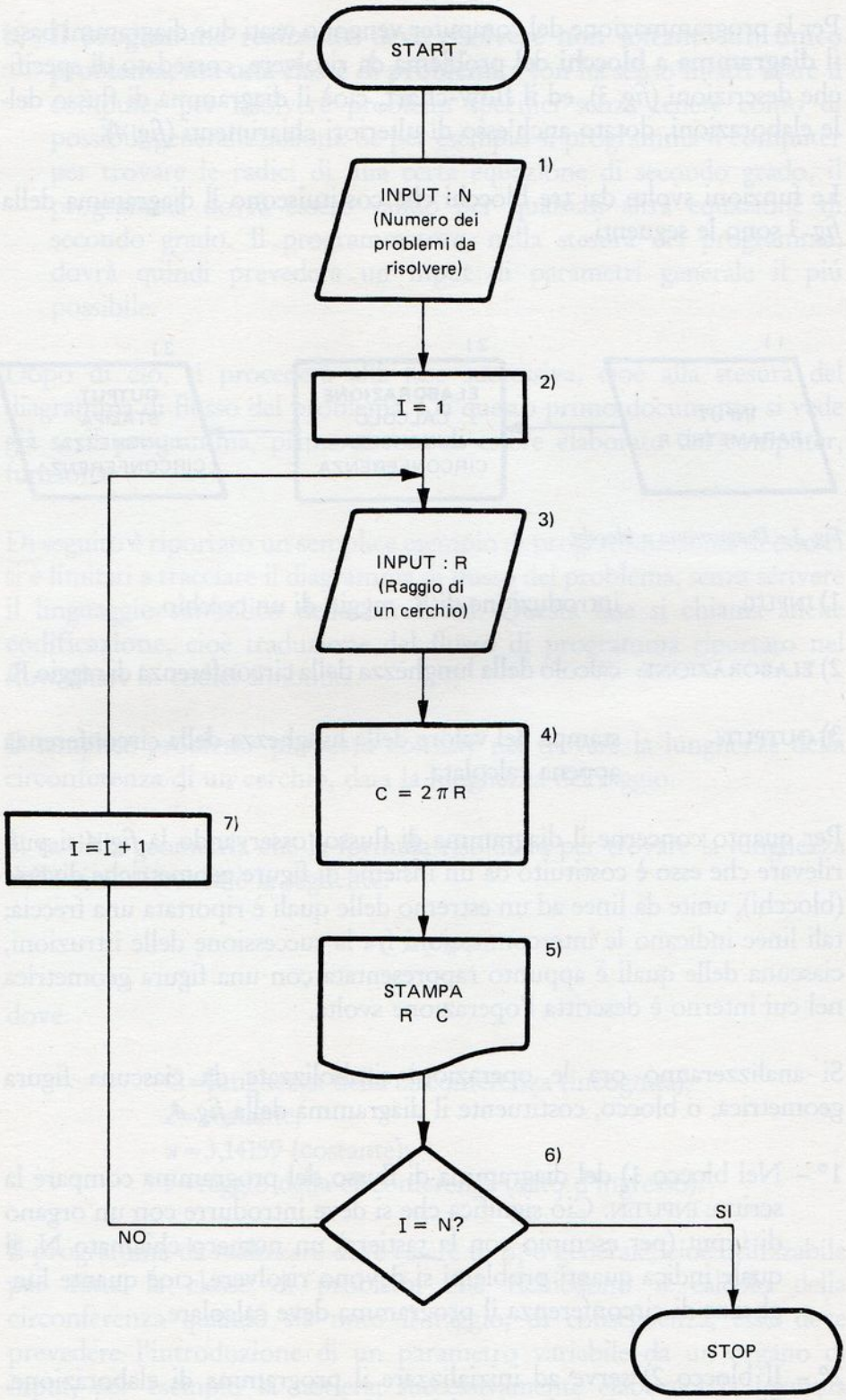



*Fig. 4 – Diagramma di flusso del calcolo della circonferenza di un cerchio.*

poi il valore 2, poi il valore 3 fino ad arrivare a N (numero dei problemi da risolvere).

3° – Nel blocco 3) deve essere fatto entrare (sempre da un organo di input) il parametro numerico R (lettera usata per indicare il raggio di un cerchio). Il numero R (sempre positivo) va introdotto in formato particolare, secondo la rappresentazione anglosassone, ossia usando il punto al posto della virgola.

Per esempio, un raggio lungo 4,5 cm si scriverà:

4.5

**L'elaborazione**

4° – Il blocco 4) esegue l'elaborazione aritmetica del programma. Il valore del calcolo dell'espressione:

$$2 \cdot \pi \cdot R,$$

dove $\pi = 3{,}14159$, viene assegnato alla variabile di nome C (lunghezza della circonferenza di raggio R).

**L'output**

5° – Il blocco 5), come già precedentemente visto, è un simbolo di stampa. Con questa istruzione vengono stampati su carta (sulla stampante) o visualizzati sul monitor il parametro di input R ed il risultato dell'elaborazione C; di solito, accanto o prima del risultato di una qualsiasi elaborazione è utile riportare anche i dati o parametri d'ingresso (in questo caso R), per una eventuale verifica.

**Il ciclo di programma**

6° – Il blocco 6) è un simbolo di "diramazione condizionale". Se il valore del contatore I, in un certo punto del flusso di elaborazione, è uguale al numero N, il programma si "dirama" al punto d'arresto STOP; se invece è diverso da N, si "dirama" al blocco 7), dove il contatore indice stesso si aggiorna di una unità. In altri termini, ciò significa che, se il programma ha eseguito N calcoli di circonferenze, si arresta, altrimenti prosegue ad elaborare finché non ha finito.

7° – Il blocco 7), come si è visto, serve ad aggiungere 1 al contatore I; dopo il blocco 7) il programma ritorna al blocco 3): si dice anche "loop al punto 3".

Il **loop** (ciclo) è un elemento di fondamentale importanza nel flow-chart. Esso, come si avrà modo di constatare in seguito, consiste in una successione di istruzioni che devono essere

ripetutamente eseguite dal calcolatore sino a quando non si verifica una condizione che altera l'ordine di esecuzione di questa successione di istruzioni costituenti il loop stesso.

## 3. – Analisi e programmazione

Il programmatore è la figura centrale di un centro di elaborazione dati; la sua funzione è determinante affinché i programmi possano essere elaborati dal calcolatore.

Non altrettanto si può dire del programmatore come figura di un sistema informativo, perché in questo contesto ci sono ben altre funzioni indispensabili allo svolgimento del flusso informativo: per esempio, quella dell'analista di sistema, il quale ha il compito non facile di recepire le esigenze informative di una varietà piuttosto vasta di utenti e di trasferirle in modo semplificato alla programmazione.

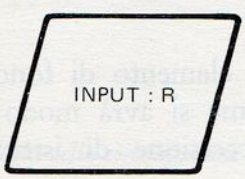
**Il diagramma a blocchi del problema da risolvere**

Comunque, il programmatore riceve dall'analista uno schema piuttosto sintetico del problema da risolvere, detto **diagramma a blocchi** (del tipo di quello riportato nella *fig. 3*), in cui sono riportate delle descrizioni riguardanti la natura dei dati d'ingresso, il funzionamento dell'algoritmo di elaborazione, l'entità e la rappresentazione dei risultati da riportare sulla stampante, ecc.

**Il diagramma di flusso del problema da risolvere**

Il compito del programmatore è quello di schematizzare il diagramma a blocchi in un ulteriore e piú dettagliato diagramma: il flow-chart, il quale altro non è che **una rappresentazione grafico-simbolica piú dettagliata del diagramma a blocchi** (come si è visto nel semplice problema del calcolo di una circonferenza).

Questa rappresentazione però deve essere formulata in termini tali da poter essere tradotta nel linguaggio simbolico di programmazione; per esempio, il simbolo sotto riportato:



Pagina mancante

## 4. - Il diagramma di flusso

L'analisi e la programmazione, come si è visto, rappresentano due fasi veramente importanti nell'elaborazione di sistemi informativi, almeno di quelli di una certa consistenza, come le elaborazioni amministrative e gestionali di un'azienda.

Per le piccole procedure e per i programmi brevi la diagrammazione a blocchi non è indispensabile, in quanto essa si può ridurre praticamente al diagramma di flusso. Eventuali descrizioni e commenti sono riportati nel flow-chart o in moduli separati.

D'altra parte, i moderni linguaggi di programmazione consentono di raffigurare lo stesso diagramma di flusso nel modulo contenente le istruzioni simboliche. Innanzitutto si possono scrivere delle istruzioni (cosiddette di commento) che consentono, per il modo in cui sono scritte, di "commentare" e descrivere il passo di programma elencato nelle righe successive: il commento può essere messo a guisa di intestazione e diventa cosí uno strumento di controllo (un flow-chart inserito tra le istruzioni programmate).

Per esempio, il programma visto precedentemente che calcola la lunghezza di una circonferenza, può essere scritto sul foglio di programmazione nel seguente modo:

**Le "istruzioni" di commento**

```
C*************************************************
C***PROGRAMMA CALCOLO CIRCONFERENZA CERCHIO***
C*************************************************
C****      FASE 1:INTRODUZIONE RAGGIO        ***
C*************************************************
```

(introdurre istruzioni di INPUT)

```
C*************************************************
C***  FASE 2:CALCOLO LUNGH. CIRCONFERENZA  ***
C*************************************************
```

(introdurre istruzioni di CALCOLO)

```
C**********************************************
C***          FASE 3:STAMPA RISULTATO          ***
C**********************************************
```

(introdurre istruzioni di OUTPUT).

In questo semplice esempio, praticamente le tre fasi di analisi, programmazione, codificazione in linguaggio simbolico e le relative descrizioni sono state unificate in un solo documento: nel MODULO DI CODIFICA.

Può questa essere considerata una regola generale? La risposta dipende dalla complessità del problema da elaborare e dal tipo di linguaggio simbolico che si usa.

Linguaggi simbolici evoluti come il FORTRAN, il BASIC, il COBOL consentono questo tipo di approccio unificato, ma non sempre.

Per esempio, nel linguaggio COBOL che si utilizza quasi sempre per elaborazioni di grosse procedure gestionali, dove sono richieste grandi quantità di dati preregistrati su memorie ausiliarie (dischi e nastri magnetici), e dove il flusso operativo degli organi di input/output è piuttosto complesso, la struttura a commenti del tipo esaminato prima non è piú sufficiente. In tal caso occorre l'ausilio di altri supporti costruiti a monte, come diagrammi a blocchi con specifiche documentate, diagrammi di flusso che descrivono in modo piú dettagliato i diagrammi a blocchi, ecc.

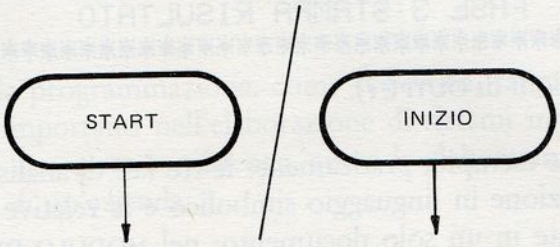
I file

Inoltre la grossa procedura richiede l'utilizzo dei FILE (archivi di dati organizzati e registrati nelle memorie ausiliarie di massa), che vengono rappresentati con appositi simboli, come si vedrà piú avanti in questa stessa lezione.

Oltre a tali simboli, vengono utilizzate nei diagrammi altre forme geometriche con significati diversi, come ad esempio: introduzione di un file dati da lettore di schede, introduzione di dati mediante terminali video, ecc.

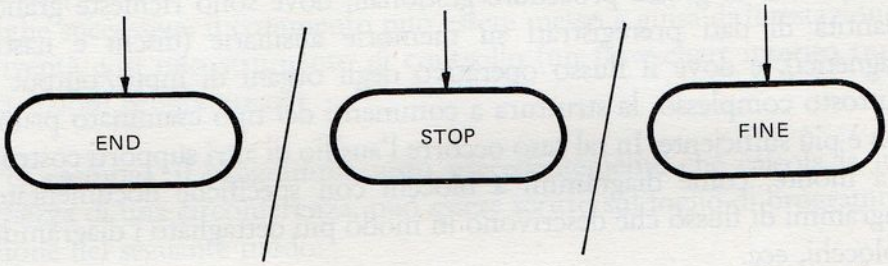
Il significato di tutti questi simboli può essere chiarito soltanto attraverso una documentazione grafica descrittiva, che può essere visualizzata sullo stesso supporto visivo in cui sono riportate le istruzioni di programma, cioè sul diagramma di flusso.

I principali simboli grafici utilizzati nel flow-chart sono rappresentati di seguito.

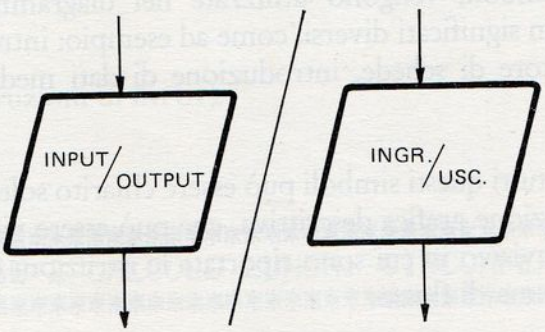



Questo simbolo si utilizza all'inizio del programma di elaborazione ed indica **il punto da cui parte il flusso delle istruzioni.**

Ad esso non corrisponde nessuna istruzione simbolica particolare (non c'è un'istruzione **START**); al massimo si possono scrivere istruzioni di commento.

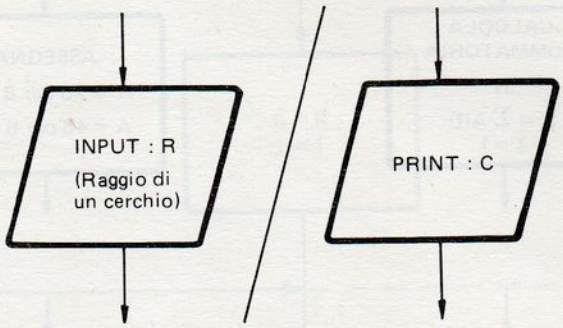



Questo simbolo indica l'**arresto**, la **fine** dell'elaborazione; ad esso corrispondono diverse istruzioni di arresto (a seconda del tipo di linguaggio di programmazione): **STOP, END.**



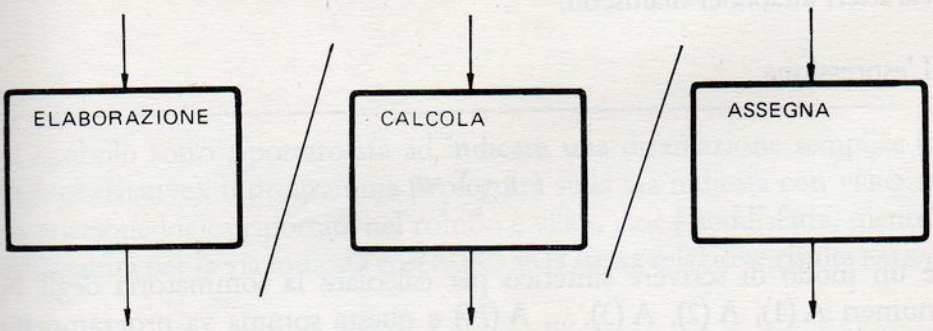

Questo simbolo rappresenta **una fase di input/output**, ossia un'operazione generica di ingresso o di uscita dei dati.

*Esempio*



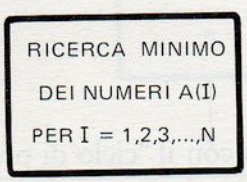

In questi piccoli parallelogrammi sono riportate, per maggiore chiarezza, delle descrizioni in merito a quello che si sta facendo. Se, ad esempio, occorre introdurre mediante la tastiera un parametro R, si scriverà INPUT: R; se si deve visualizzare sullo schermo o sul monitor il valore della variabile C, si scriverà PRINT: C, e cosí via.

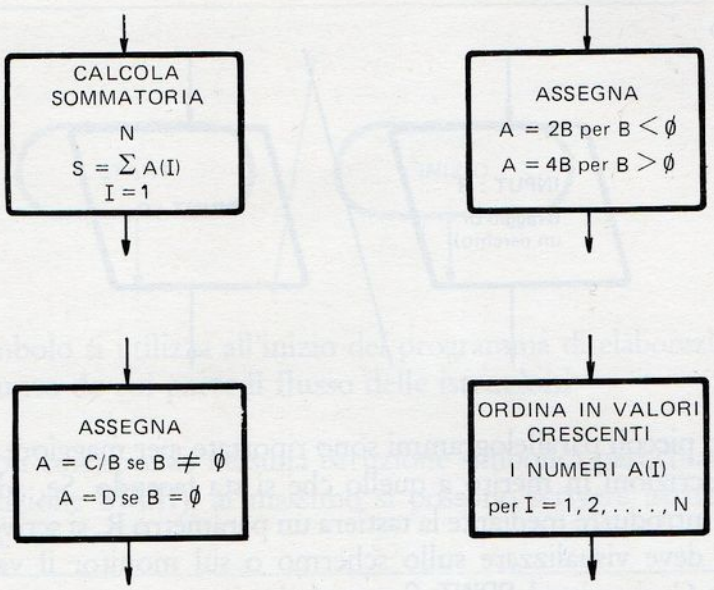
In genere, le lettere o i gruppi di lettere che si usano per rappresentare certi parametri o dati sono gli stessi utilizzati successivamente nel linguaggio simbolico delle istruzioni. Spesso vengono adottati anche simboli diversi per indicare l'apparecchiatura periferica d'ingresso o d'uscita impiegata (stampante, unità a nastro, a disco, ecc.).




E' questo il simbolo grafico piú usato per indicare che **si sta eseguendo un'operazione** di calcolo logico-aritmetico o, piú genericamente, **un'azione di elaborazione.**

*Esempi*






Come si vede, in molte elaborazioni vengono usati simboli matematici particolari, come $\Sigma$, $\pi$, ecc.

Questi simboli sono adottati per brevità di rappresentazione: quando si scriverà il programma nel linguaggio di programmazione prescelto, al posto di $\pi$ (pi-greco) si userà 3.14159, al posto di **r** si scriverà R, perché sulla tastiera dei piú comuni personal computer sono riportati solo i caratteri alfabetici maiuscoli.

L'espressione

$$S = \sum_{I=1}^{N} A(I)$$

è un modo di scrivere sintetico per calcolare la sommatoria degli N numeri A (1), A (2), A (3), ..., A (N) e questa somma va programmata utilizzando un particolare deposito di memoria (di nome S) su cui si accumulano via via i vari dati A (I) per I=1, 2, ..., N che sono anch'essi depositati in memoria nell'ordine A (1), A (2), ..., A (N).

Il calcolo

$$S = \sum_{I=1}^{N} A(I)$$

si realizza con il "ciclo di programma" riportato nella *fig. 5*.

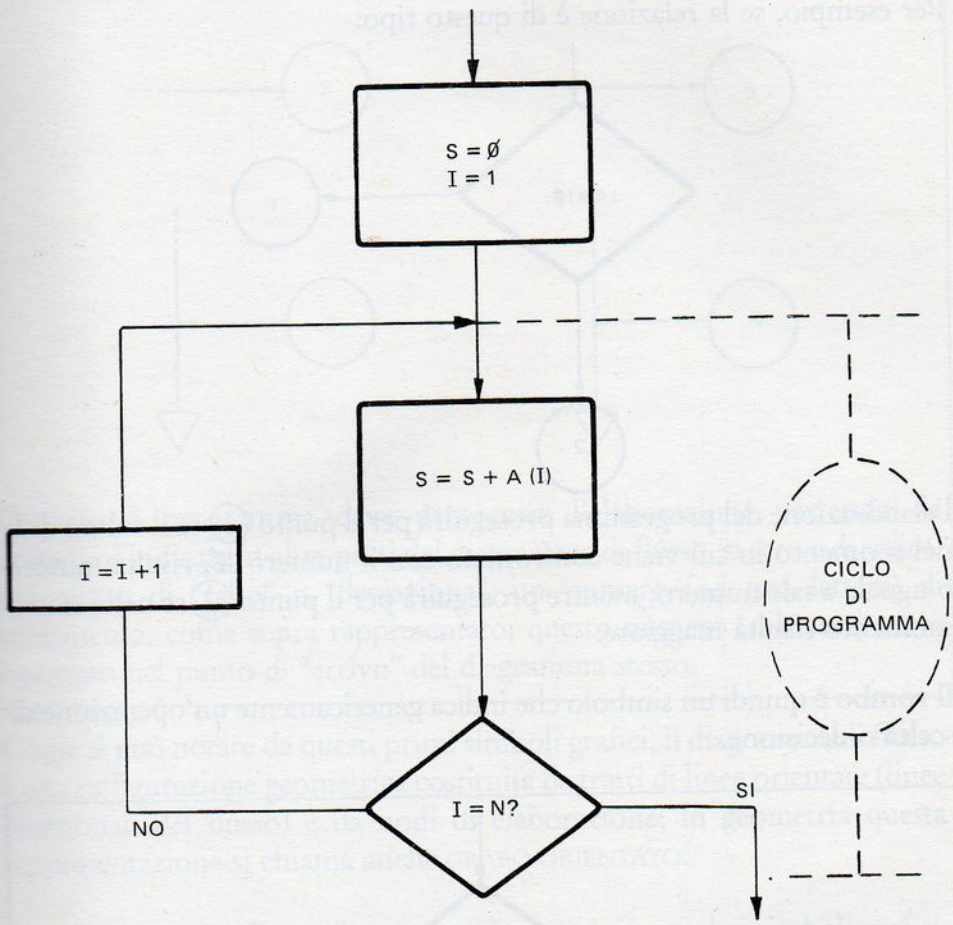
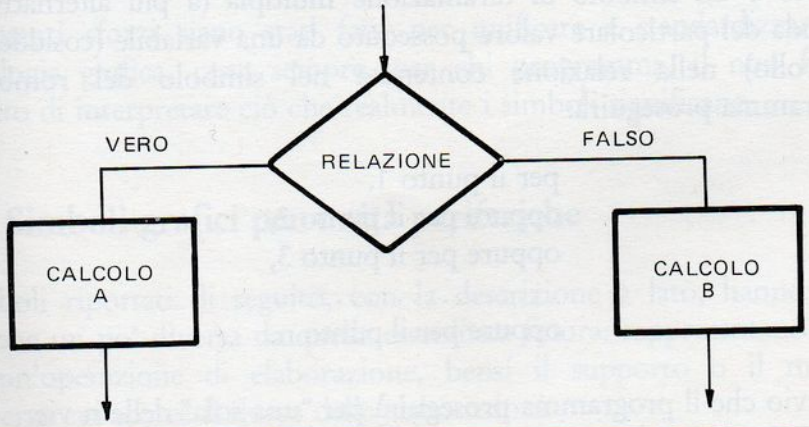


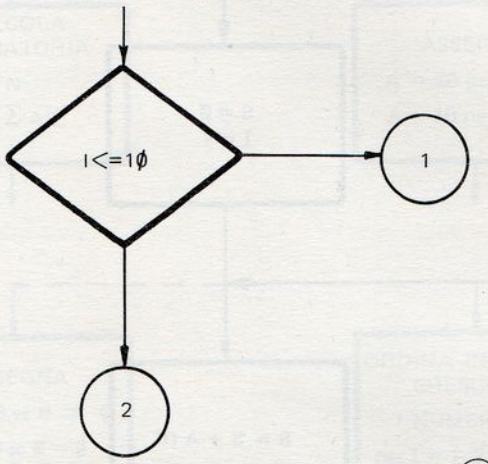

*Fig. 5 – Ciclo di programma per il calcolo di* $S = \sum_{I=1}^{N} A(I)$.

**La diramazione semplice**

Il simbolo sotto riportato sta ad indicare una diramazione semplice (a due alternative): il programma proseguirà sulla via indicata con VERO se la relazione logica riportata nel rombo è VERA, cioè è soddisfatta, mentre proseguirà per la via indicata con FALSO se la stessa relazione risulta FALSA.

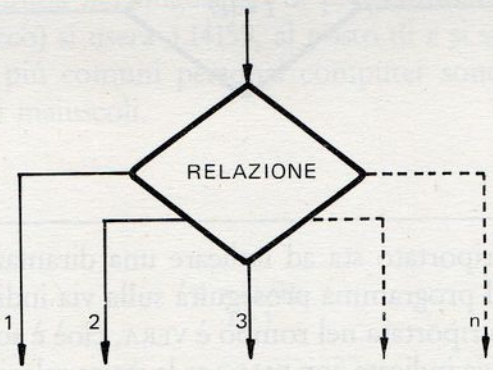


Per esempio, se la relazione è di questo tipo:




l'elaborazione del programma proseguirà per il punto (1) se il valore di I, nel momento in cui viene confrontato con il numero 1Ø, risulta minore o uguale a tale numero, mentre proseguirà per il punto (2) se nello stesso momento risulta maggiore.

Il **rombo** è quindi un simbolo che indica genericamente un'operazione di scelta o decisione.

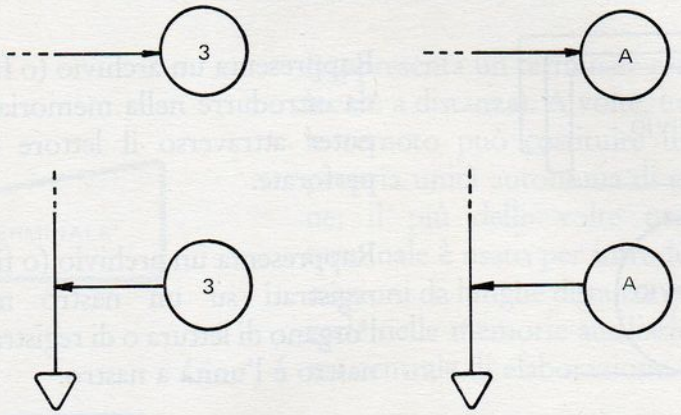



**La diramazione multipla**

Questo è un simbolo di diramazione multipla (a piú alternative): a seconda del particolare valore posseduto da una variabile (cosiddetta di controllo) nella relazione contenuta nel simbolo del rombo, il programma proseguirà:

per il punto 1,
oppure per il punto 2,
oppure per il punto 3,
. . . . . .
oppure per il punto n.

E' ovvio che il programma proseguirà per "una sola" delle **n** vie.




Quando il programma deve proseguire l'elaborazione partendo da istruzioni indicate in altre parti del diagramma di flusso, si scrive di solito, nel punto di "salto" o "deviazione", un numero (od una lettera) di riferimento, come sopra rappresentato; questo numero (o lettera) verrà riportato nel punto di "arrivo" del diagramma stesso.

**Il punto di "salto" e il punto di "arrivo"**

Come si può notare da questi primi simboli grafici, il diagramma di flusso è una raffigurazione geometrica costituita da tratti di linee orientate (linee direzionali del flusso) e da nodi di elaborazione; in geometria questa rappresentazione si chiama anche GRAFO ORIENTATO.

Oltre a queste due figure (linee e nodi) sono riportati altri simboli grafici, caratteristici di un sistema di elaborazione, che saranno descritti brevemente nel prossimo paragrafo.

E' bene notare che il flow-chart va definito con la massima chiarezza possibile; se ci sono equivoci nell'interpretazione di certi simboli, occorre eliminarli, ricorrendo magari a descrizioni separate.

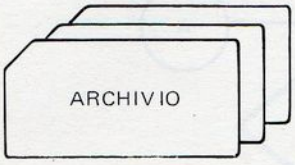
Per quanti sforzi siano stati fatti per unificare e standardizzare la simbologia grafica, resta sempre, per chi programma, il non facile compito di interpretare ciò che realmente i simboli significano.

**Standard dei simboli grafici**

## 4.1 - Simboli grafici per unità periferiche

I simboli riportati di seguito, con la descrizione a lato, hanno una funzione un po' diversa da quella esaminata finora: rappresentano cioè non un'operazione di elaborazione, bensí il supporto o il mezzo attraverso cui scorre il flusso delle elaborazioni.

*SIMBOLI DI ARCHIVIO*



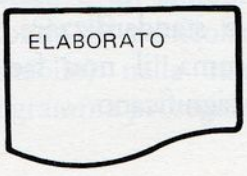

Rappresenta un archivio (o file) di dati da introdurre nella memoria del computer attraverso il lettore di schede perforate.

Rappresenta un archivio (o file) di dati registrati su un nastro magnetico; l'organo di lettura o di registrazione del nastro è l'unità a nastro.

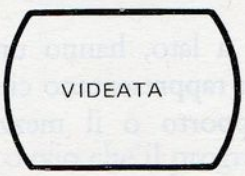
Rappresenta un archivio di dati registrati su dischi magnetici; l'organo di lettura o di registrazione dei dati è l'unità a disco.
Si ricordi che il reperimento dei dati su disco è piú immediato che non su nastro magnetico.

Rappresenta una banda di carta perforata (dati codificati secondo un codice di perforazione, analogamente alla scheda perforata); l'organo di input che legge le informazioni riportate sulla banda è il lettore di banda o FOTOLETTORE.

*SIMBOLI DI STAMPA*

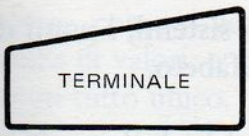



Rappresenta un foglio di stampa su cui vengono riportati normalmente i risultati dell'elaborazione; si chiama anche ELABORATO, cioè prospetto stampato dei risultati. L'organo di stampa è la stampante.



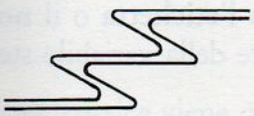

Rappresenta lo schermo televisivo di un terminale-video o monitor. Con questo strumento è possibile visualizzare e controllare sia i dati di ingresso, sia i risultati dell'elaborazione.

*SIMBOLI DI ELABORAZIONE REMOTA*




Rappresenta un terminale remoto (collocato a distanza). A volte, un terminale remoto può costituire una vera e propria unità autonoma di elaborazione; il piú delle volte però questo terminale è usato per introdurre informazioni da lunghe distanze e memorizzarle nelle memorie ausiliarie del sistema centrale di elaborazione.

Rappresenta lo schermo di un terminale video; è usato per indicare l'introduzione di dati da un terminale remoto (remote job entry).



Rappresenta un cavo telefonico per comunicazioni a distanza (data communication).

## 5. - Nozioni sull'organizzazione della memoria del microcomputer

Nella lezione precedente si sono visti alcuni aspetti particolari dell'unità di memoria principale del microcomputer, nell'intento di mettere a fuoco nel modo piú immediato alcuni concetti che sono certamente da considerare fondamentali per la comprensione del funzionamento del microcalcolatore stesso.

Si vedrà ora come vengono memorizzati in tale memoria i dati variabili e come essa è organizzata.

Le variabili svolgono il compito di portatrici di valori: se un valore viene assegnato ad una variabile, rimane associato ad essa sino a quando non viene assegnato un nuovo valore alla variabile stessa.

Le variabili possono essere di tipo numerico (2, 1Ø.5, Ø.ØØ8, ecc.), logico (vero, falso) e alfanumerico ("Scuola Radio Elettra", "I+ 1ØN", ecc.).

A seconda del sistema di elaborazione del linguaggio di programmazione, i nomi delle variabili possono essere costituiti da parole mnemoniche di due o piú lettere e numeri: il linguaggio BASIC normalmente riserva una oppure due lettere e numeri per i nomi delle variabili.

**In base a una regola generale, valida per tutti i sistemi, i nomi delle variabili iniziano sempre con una lettera dell'alfabeto.**

Le variabili vengono memorizzate nella memoria del calcolatore in zone ben definite e predeterminate dal sistema operativo (quest'ultimo è costituito dall'insieme delle istruzioni di servizio, aventi lo scopo di gestire i programmi nel calcolatore).

Per esempio, la variabile di nome I occuperà la zona chiamata I, la variabile di nome S occuperà una zona denominata S e cosí via.

La memoria del microcomputer si può pertanto configurare come una cassettiera: sulla facciata di ogni cassetto apparirà l'**etichetta** o il **nome** della variabile e dentro il cassetto vi sarà il valore della variabile stessa, come mostrato nella *fig. 6-a*.

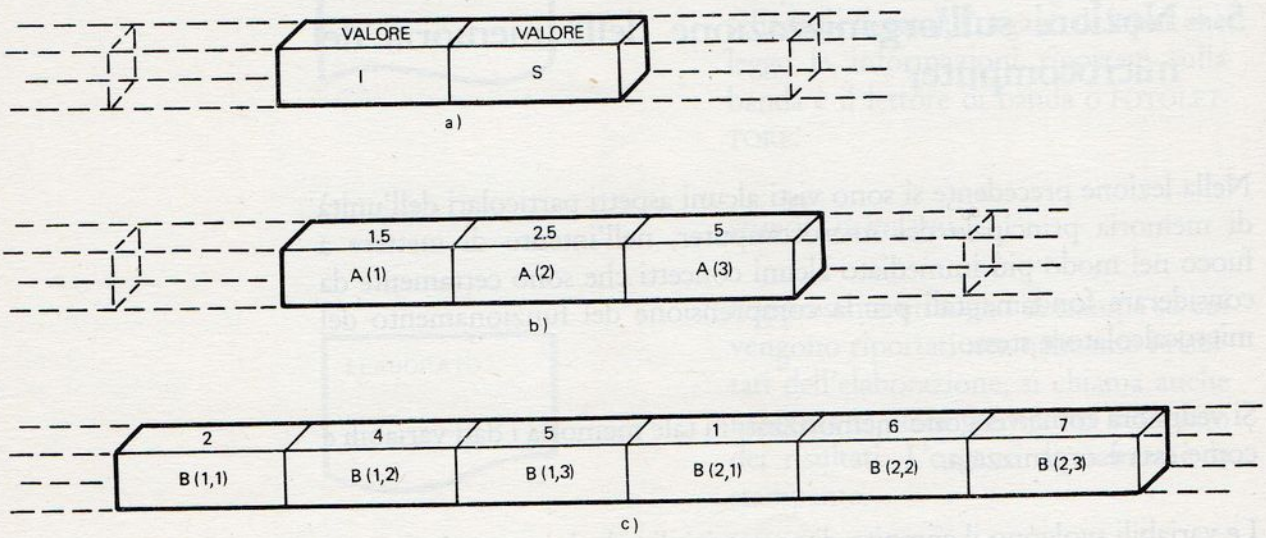



*Fig. 6 – Memorizzazione dei dati variabili semplici (a), dei vettori (b) e delle matrici (c).*

Anche le variabili con indice (vettori e matrici) sono memorizzate in una zona definita della memoria.

La **variabile con indice** è una variabile formata da un insieme di valori, anziché da un unico valore. Ad esempio, la variabile ad un indice costituita dal vettore A (I), formato da tre numeri reali A (1)=1.5, A (2)=2.5, A (3)=5, viene memorizzata come indicato nella *fig. 6-b.*

Il vettore, come si vedrà in una prossima lezione, è costituito da una sequenza di valori, e rappresenta un elenco ordinato di dati da trattare come un tutto unico.

Come secondo esempio di variabile con indici si può considerare la matrice B (I, J) di due righe (I=2) e tre colonne (J=3), costituita dai seguenti numeri interi:

B(1,1)=2
B(1,2)=4
B(1,3)=5
B(2,1)=1
B(2,2)=6
B(2,3)=7.

Questa matrice viene memorizzata nel modo rappresentato nella *fig. 6-c*; come si può osservare, essa è una variabile a due indici, in quanto l'elemento B è individuato dalle coppie di valori indicate fra parentesi.

A questo punto è opportuno precisare che **la matrice** (ente algebrico) **non è altro che un modo particolare di indicare con un simbolo unico un insieme di numeri organizzati in una tabella costituita da righe e colonne.**

Ecco come viene espressa in notazione grafica la matrice 2×3 dell'esempio visto, con i valori dei suoi elementi definiti esplicitamente.

$$B=\underbrace{\left.\begin{bmatrix}2&4&5\\1&6&7\end{bmatrix}\right\}\text{righe}}_{\text{colonne}}$$

L'algebra matriciale è uno dei piú efficienti metodi di calcolo e trova applicazione in svariati settori della tecnica e della scienza; inoltre, questo tipo di tecnica matematica è particolarmente adatto ad essere usato nel calcolatore.

Come si è potuto constatare dagli esempi considerati, l'organizzazione dei dati nella memoria principale del computer è di tipo "sequenziale" nel

# Tipi di memorie

Le memorie utilizzate nel microcomputer sono costituite da strutture elementari di semiconduttori, denominate CELLE o LOCAZIONI (capaci di memorizzare un'informazione binaria), ed organizzate in forma di tabella o matrice, ossia disposte in righe e colonne.

Come già precisato nella precedente lezione, i due tipi di memorie piú comunemente usati sono le RAM e le ROM.

Nella *fig. 7* è rappresentato lo schema a blocchi di una RAM (memoria che può essere letta e scritta durante il funzionamento); come si può osservare, è sempre presente un certo numero di **ingressi di indirizzo** che, unitamente ai decodificatori di riga e di colonna, consentono di selezionare e quindi localizzare ogni informazione memorizzata.

Oltre ai suddetti ingressi vi sono un **ingresso scrittura** per la memorizzazione dei dati, un'**uscita lettura** per il prelievo dei dati memorizzati, un **ingresso lettura-scrittura** L/S che abilita la memoria all'una o all'altra delle due funzioni ed infine un **ingresso di selezione** (chip select: CS) che permette di selezionare la RAM desiderata qualora nel sistema ne sia impiegata piú di una.

Quando il microcomputer riceve l'istruzione per la localizzazione di una parola nella RAM, pone gli indirizzi di tale parola sul bus degli indirizzi. Localizzato il dato nella memoria RAM, questa lo invia sul bus dei dati. Il microcomputer può allora trasferire quel dato verso l'esterno del sistema attraverso il circuito ingresso/uscita, oppure elaborarlo e poi inviare il risultato nella RAM (sempre attraverso il bus dei dati) nella medesima posizione oppure in una posizione diversa.

La capacità delle RAM (come pure quella delle ROM) è indicata in unità K, il cui valore equivale a 1.024 byte (1 K = 1.024 byte); per esempio, una memoria da 16 K equivale a 16.384 byte.

Le RAM piú comuni hanno una capacità che va da 64 byte a 16.384 byte.

Nella memoria ROM (che può essere solamente letta) sono memorizzate le istruzioni costituenti il programma operativo del microcomputer.

Ogni tipo di microcomputer è quindi progettato per riconoscere un preciso gruppo di istruzioni, fornite dal costruttore, le quali costituiscono il suo linguaggio. Le ROM che si utilizzano in pratica hanno una capacità compresa fra 256 byte e 65.536 byte (64 K).

L'organizzazione tipica di una memoria ROM è simile a quella delle memorie RAM; anche in questo caso sono presenti ingressi di indirizzo ed uno o piú ingressi di selezione (CS), che consentono di selezionare la ROM desiderata qualora nel sistema ne sia impiegata piú di una. Naturalmente nella memoria ROM non vi sono piú l'ingresso dati (scrittura) e l'ingresso L/S.

Oltre ai due tipi di memoria descritti, vengono frequentemente utilizzate nei microcomputer le memorie PROM e EPROM.

Le PROM (Programmable ROM, ossia ROM programmabile) sono del tutto uguali, dal punto di vista funzionale, alle memoria ROM, ma presentano, rispetto a queste ultime, la possibilità di **essere programmate una sola volta dall'utente prima di essere utilizzate.** La programmazione si realizza mediante un'apposita apparecchiatura, che provoca nella matrice di memoria l'interruzione di microfusibili oppure la fusione di giunzioni di semiconduttori, in modo da conseguire la desiderata configurazione logica alle uscite durante la lettura. Queste memorie consentono quindi all'utente del microcomputer di programmarle in proprio, inserendo informazioni non previste dal costruttore.

Le memorie EPROM (Erasable Programmable ROM, ovvero memorie ROM programmabili e cancellabili) presentano la caratteristica di **poter essere cancellate e nuovamente riscritte ogni volta che l'utente lo desidera.** La cancellazione del contenuto avviene tramite raggi ultravioletti, che giungono all'interno dell'integrato contenente la memoria EPROM attraverso una piccola finestra di quarzo, materiale che lascia passare agevolmente i raggi ultravioletti (*fig. 8*).

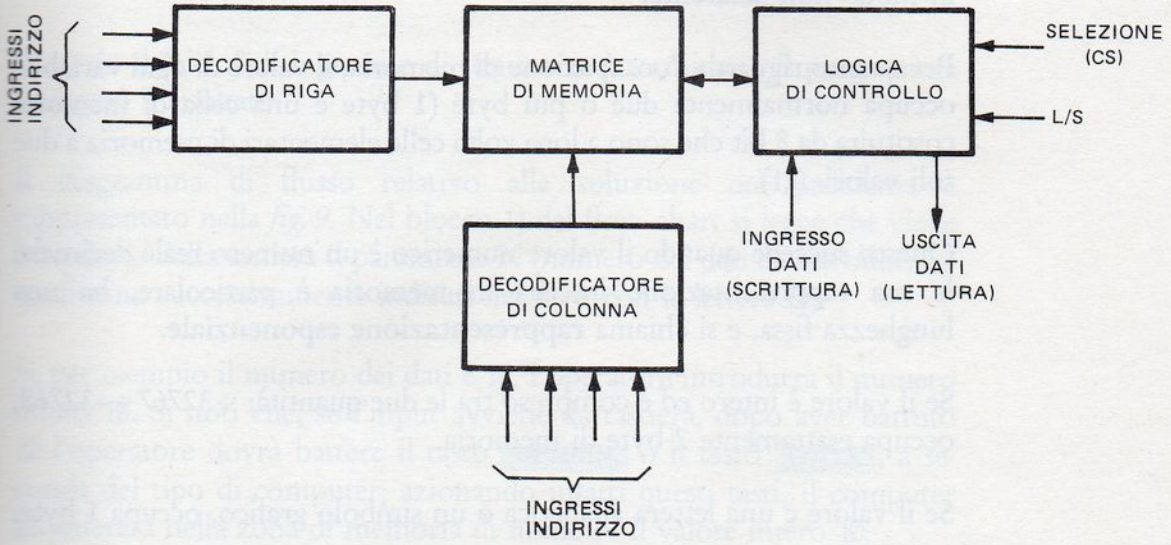



*Fig. 7 – Organizzazione di una tipica memoria RAM.*

*Fig. 8 – Memoria EPROM.*

caso di vettori e matrici, cioè di dati forniti di indici; la sequenza è determinata dai valori degli indici stessi. Nel caso invece di variabili semplici come I, S, viste prima, queste possono essere memorizzate anche in modo non adiacente.

Per quanto riguarda l'occupazione di memoria, il valore di ogni variabile occupa normalmente due o piú byte (1 byte è una cella di memoria costituita da 8 bit che sono a loro volta celle elementari di memoria a due soli valori: 0,1).

Questo succede quando il valore numerico è un numero reale decimale: la sua rappresentazione interna in memoria è particolare, ha una lunghezza fissa, e si chiama **rappresentazione esponenziale.**

Se il valore è intero ed è compreso tra le due quantità: +32767 ÷ -32768, occupa esattamente 2 byte di memoria.

Se il valore è una lettera alfabetica o un simbolo grafico, occupa 1 byte.

Anche i nomi delle variabili occupano 1 o piú byte di memoria.

## 6. - Esempi di programmazione

Negli esempi di programmazione che seguono, si prenderà in considerazione la soluzione di tre semplici problemi.

***1° problema*** - *Calcolo della somma di N numeri.*

Il problema in questione consiste nel far calcolare dal computer la somma di N numeri reali (interi o decimali), introdotti da una unità d'ingresso (per esempio, dalla tastiera).

In questo semplice problema viene considerato un solo parametro d'ingresso (N, cioè il numero dei dati da sommare) e come dati d'ingresso si assumono i valori numerici dei numeri da sommare.

Il programma quindi inizierà con l'introduzione del parametro N (numero dei dati da sommare); successivamente proseguirà con l'introduzione di un dato per volta: ci saranno tanti ingressi quanti sono i dati da sommare (cioè N ingressi).

Contemporaneamente all'ingresso di ogni dato, il programma dovrà anche provvedere ad accumulare il nuovo dato al numero introdotto precedentemente; a questo scopo si potrà ricorrere ad un deposito accumulatore (di nome S), in cui via via si sommeranno i vari numeri.

Quando il "ciclo" di ingressi e di somme sarà terminato, il programma stamperà il risultato.

Il diagramma di flusso relativo alla soluzione del problema è rappresentato nella *fig. 9*. Nel blocco 1) del flow-chart si legge che viene introdotto dalla tastiera il parametro N (numero dei dati da introdurre, i quali sono successivamente indicati nel flow con la lettera D).

Se per esempio il numero dei dati è 1Ø, l'operatore introdurrà il numero intero 1Ø. Si noti che, se l'input avviene da tastiera, dopo aver battuto 1Ø l'operatore dovrà battere il tasto **RETURN** o il tasto **ENTER**, a seconda del tipo di computer; azionando infatti questi tasti, il computer incamererà nella zona di memoria di nome N il valore intero 1Ø.

Nel blocco 2) viene assegnato il valore 1 al contatore indice di nome I ed il valore Ø (questo carattere indica il numero zero, e viene usato per non confonderlo con la lettera O) alla variabile numerica di nome S.

Perché si assegna 1 all'indice I? Perché il contatore, che serve per contare quanti ingressi e quante somme il computer deve eseguire, deve partire con il numero 1.

Il deposito accumulatore della somma (di nome S) dovrà invece avere, all'inizio della sommatoria, il valore Ø.

Con il blocco 3) si entra nel "ciclo" di programma. Come si vede dal flow-chart, questo ciclo comprende i blocchi 3), 4), 5), 6) e viene ripetuto esattamente N volte, cioè tante volte quanti sono i dati da introdurre.

La variabile che controlla questo ciclo è quella che si è precedentemente indicata con la lettera I: il ciclo inizia con l'indice I a valore 1 e termina con l'indice I a valore N.

Nel blocco 3) si legge che viene introdotto un dato numerico; questo dato va introdotto dalla tastiera in formato reale (cioè in forma decimale o intera). Anche in questo caso, dopo aver battuto il numero, l'operatore dovrà premere il tasto **RETURN** o il tasto **ENTER**. Si ricordi che il

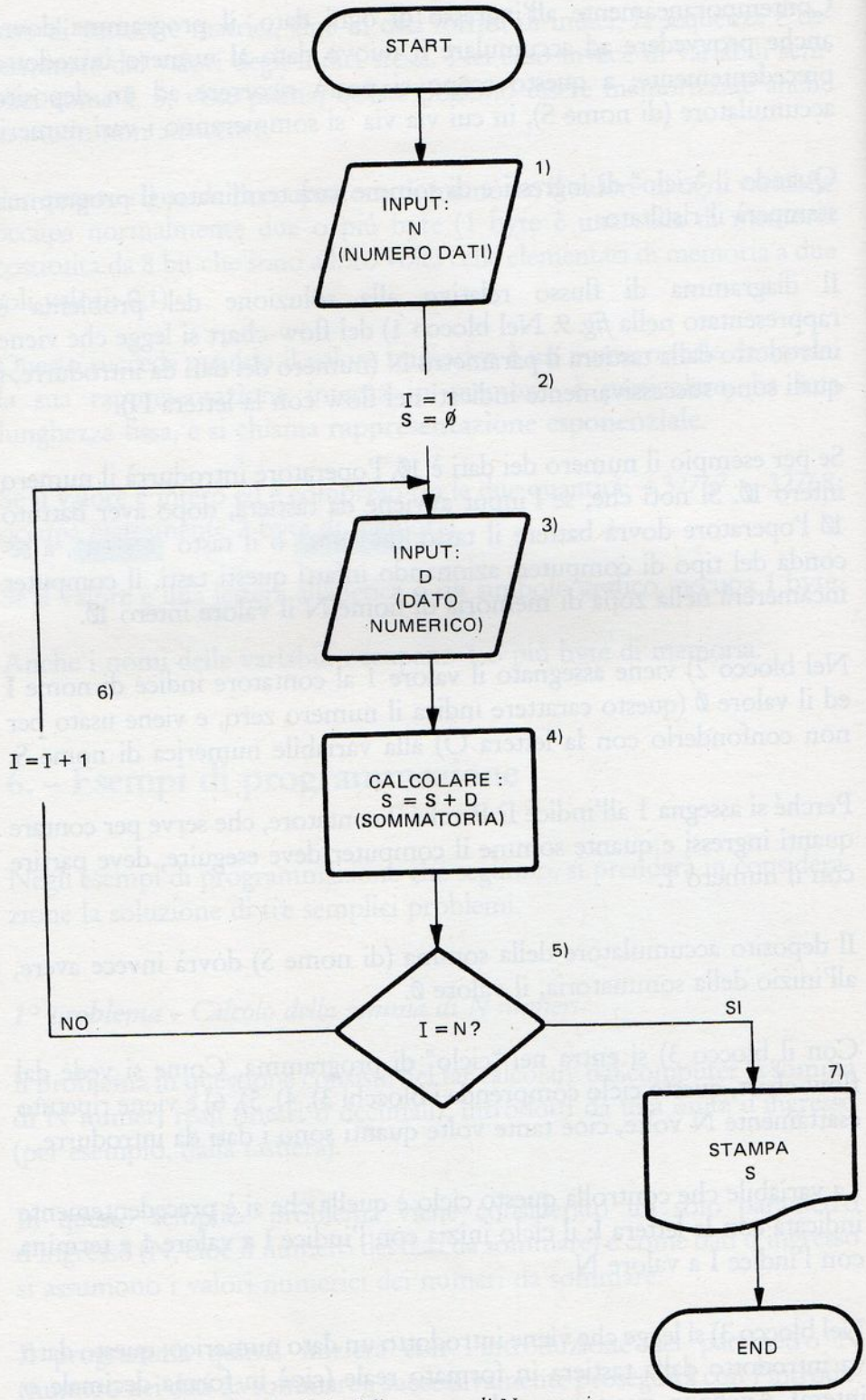



*Fig. 9 – Diagramma di flusso per il calcolo della somma di N numeri.*

numero, se è decimale, dovrà essere battuto con il punto anziché con la virgola.

Il dato generico che si introduce risiederà temporaneamente (fino a che non verrà cancellato dal successivo numero) nella zona di memoria del computer di nome D.

Nel blocco 4) viene calcolata la sommatoria (o l'accumulo) dei dati che via via si immettono nel computer.

Se per esempio si introducono i seguenti tre numeri (N=3):

5
7.1
8.25

| | | |
|---|---|---|
| per I=1 | S avrà il valore | Ø+5=5 |
| per I=2 | S avrà il valore | 5+7.1=12.1 |
| per I=3 | S avrà il valore | 12.1+8.25=2Ø.35 |

Quando I vale 3, la relazione logica contenuta nel blocco 5) è così formulata:

3=3?

A questa domanda il computer risponde SI; il programma "esce" quindi dal ciclo e va ad effettuare il blocco 7), dove è prevista la stampa del valore della sommatoria S.

Se il computer è dotato soltanto di video, questo risultato può essere visualizzato sullo schermo con una istruzione di stampa. Nel linguaggio BASIC, per esempio, si usa l'istruzione:

**PRINT** S

dove "PRINT" è la parola inglese che vuol dire "stampa" e S è il nome del risultato.

Nel blocco 6) la variabile indice I, che come si è visto funge da controllo del ciclo, aumenta di una unità (si dice anche che I si "aggiorna" di 1). Pure nel blocco 4) la variabile sommatoria S si aggiorna dell'ultimo dato numerico introdotto.

Come si può notare, ogni formulazione del tipo:

$$I=I+1$$
$$S=S+D, \text{ ecc.}$$

assume nella programmazione un particolare significato.

Essa praticamente sta ad indicare che il valore della variabile posta a sinistra del segno = diventa un "nuovo" valore, ottenuto sommando al precedente valore della stessa variabile un altro dato o costante (nel presente esempio, il numero generico D o la costante 1).

***2° problema** – Ricerca del minimo valore numerico di N numeri reali.*

Dati N numeri reali $a_i$ per $i=1, 2, ..., N$ e supponendo che questi numeri siano già memorizzati, uno dietro l'altro, nella memoria centrale del calcolatore, occorre trovare il minore degli $a_i$ e la sua posizione nella successione $a_1, a_2, a_3, a_4, ..., a_N$.

Per esempio, nella successione 3; 4.5; 1.1; 20 il minimo valore è 1.1 che si trova nella posizione 3.

Si tratta di trovare un criterio logico (algoritmo) che consenta al calcolatore di rintracciare il valore minimo con un numero relativamente breve di confronti.

Il metodo qui proposto è quello di prendere in considerazione il primo termine della serie di numeri, confrontarlo con i successivi termini e, quando da qualche confronto esso risulta piú grande, abbandonarlo e considerare come primo termine di paragone il numero piú piccolo appena trovato.

I confronti proseguiranno da questo termine fino ad esaurire tutti i termini della successione.

Si possono indicare rispettivamente con DM i nomi dei depositi di memoria assegnati al valore minimo e con IM la posizione del minimo; nella successione citata prima come esempio sarà:

$$DM=1.1$$
$$IM=3$$

Si può indicare inoltre con I un indice (contatore) che parta dal valore iniziale 1 e, di 1 in 1, arrivi al valore N–1 (numero dei dati meno 1). L'indice I, in pratica, "conta" il numero dei confronti, che sono appunto N–1.

L'andamento del diagramma di flusso del problema in questione è riportato nella *fig. 10*. Si può verificare tale diagramma prendendo in considerazione la successione di numeri prima citata: 3; 4.5; 1.1; 20.

Dopo il blocco INIZIO si hanno quindi le tre istruzioni di assegnazione:

1) I=1
2) DM=A (I)
3) IM=I

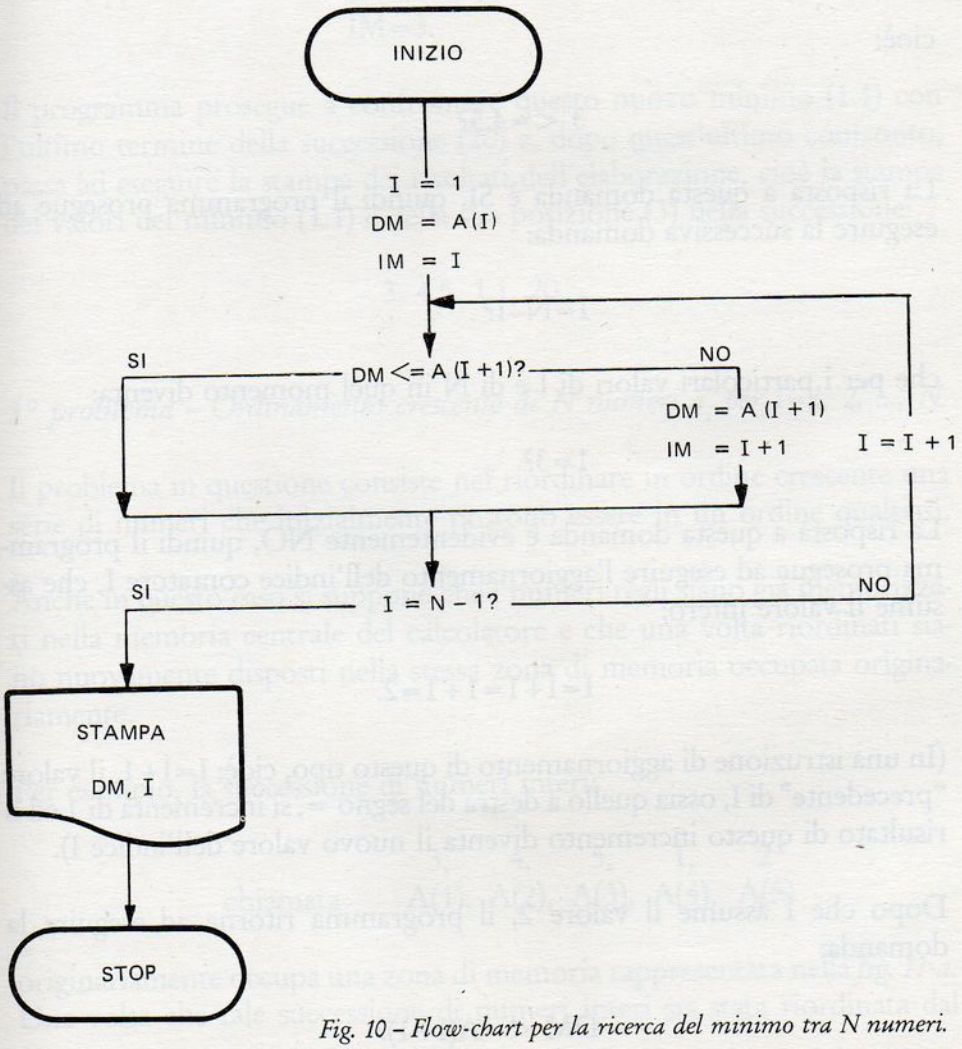



*Fig. 10 – Flow-chart per la ricerca del minimo tra N numeri.*

Queste tre istruzioni, eseguite nella sequenza 1), 2), 3) indicata sopra (cosí infatti verranno eseguite dal computer), diventano:

I=1
DM=A (1)=3
IM=I=1

Successivamente nel flow-chart si trova la domanda:

DM <=A (I+1)?

che, per i particolari valori posseduti in quel momento da DM e I, si può anche scrivere:

3 <=A (2)?

cioè:

3 <=4.5?

La risposta a questa domanda è SI, quindi il programma prosegue ad eseguire la successiva domanda:

I=N-1?

che per i particolari valori di I e di N in quel momento diventa:

1=3?

La risposta a questa domanda è evidentemente NO, quindi il programma prosegue ad eseguire l'aggiornamento dell'indice contatore I, che assume il valore intero:

I=I+1=1+1=2.

(In una istruzione di aggiornamento di questo tipo, cioè: I=I+1, il valore "precedente" di I, ossia quello a destra del segno =, si incrementa di 1 ed il risultato di questo incremento diventa il nuovo valore dell'indice I).

Dopo che I assume il valore 2, il programma ritorna ad eseguire la domanda:

DM <=A(I+1)?

Tenendo presenti i valori assunti in quel momento da DM e I, la domanda in questione viene cosí formulata:

$$3 <= A(3)?$$
$$3 <= 1.1?$$

La risposta è sempre NO ed il flow-chart indica le due istruzioni di assegnazione:

$$DM = A(I+1)$$
$$IM = I+1.$$

Queste istruzioni, per il particolare valore assunto da I in quel momento, diventano:

$$DM = A(3) = 1.1$$
$$IM = 3.$$

Il programma prosegue a confrontare questo nuovo minimo (1.1) con l'ultimo termine della successione (20) e, dopo quest'ultimo confronto, passa ad eseguire la stampa dei risultati dell'elaborazione, cioè la stampa dei valori del minimo (1.1) e della sua posizione (3) nella successione:

3, 4.5, 1.1, 20.

***3° problema*** – *Ordinamento crescente di N numeri $a_i$ per i=1, 2, ..., N.*

Il problema in questione consiste nel riordinare in ordine crescente una serie di numeri che inizialmente possono essere in un ordine qualsiasi.

Anche in questo caso si suppone che i numeri reali siano già memorizzati nella memoria centrale del calcolatore e che una volta riordinati siano nuovamente disposti nella stessa zona di memoria occupata originariamente.

Per esempio, la successione di numeri interi:

| | 3, | 4, | 5, | 1, | 2 |
|---|---|---|---|---|---|
| chiamata | A(1), | A(2), | A(3), | A(4), | A(5) |

originariamente occupa una zona di memoria rappresentata nella *fig. 11-a.* Una volta che tale successione di numeri interi sia stata riordinata dal

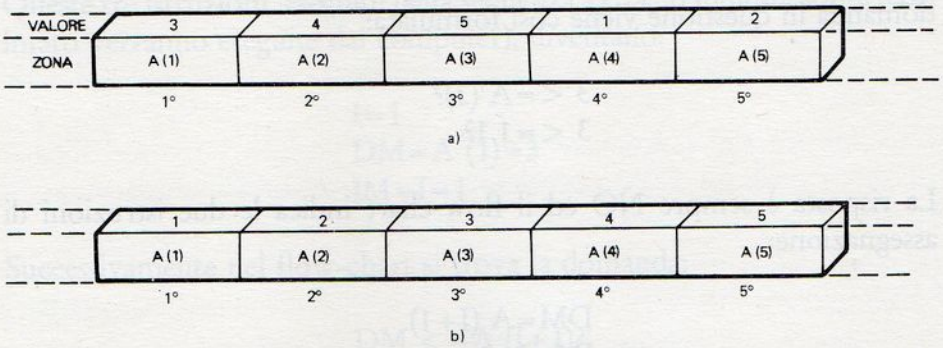



*Fig. 11 - Zone della memoria occupate dai numeri da riordinare (a) e da quelli riordinati (b).*

programma di elaborazione, dovrà occupare la stessa zona nell'ordine crescente, come mostrato nella *fig. 11-b.*

L'algoritmo ora proposto consiste nell'eseguire un certo numero di confronti tra i vari dati: quando il primo termine di paragone risulta piú grande del secondo termine, i valori dei due numeri cambiano posto.

Il numero e la successione dei confronti si possono organizzare nel seguente modo.

In una prima serie di confronti (N-1 confronti) si paragona il numero A (1) con "tutti" gli altri numeri successivi A (I), per I=2, 3, ..., N; se A (1) incontra un numero minore, scambia il suo posto con quest'ultimo.

In una seconda serie di confronti (N-2 confronti) si paragona il numero A (2) con "tutti" gli altri numeri succcessivi A (I), per I=3, 4, ..., N; se A (2) incontra un numero minore, scambia il suo contenuto (valore) con quello di quest'ultimo. Il procedimento prosegue con altre successive serie di confronti, fino ad arrivare alla penultima serie di confronti, in cui verrà eseguito un numero di confronti pari a N-(N-1)=1, cioè un solo confronto tra il penultimo e l'ultimo dato.

Il numero totale dei confronti è quindi:

$$(N-1)+(N-2)+(N-3)+.....1=\frac{N\cdot(N-1)}{2}.$$

Con l'aiuto del flow-chart della *fig. 12*, si proverà ora ad ordinare la successione di numeri interi:

3, 4, 5, 1, 2.

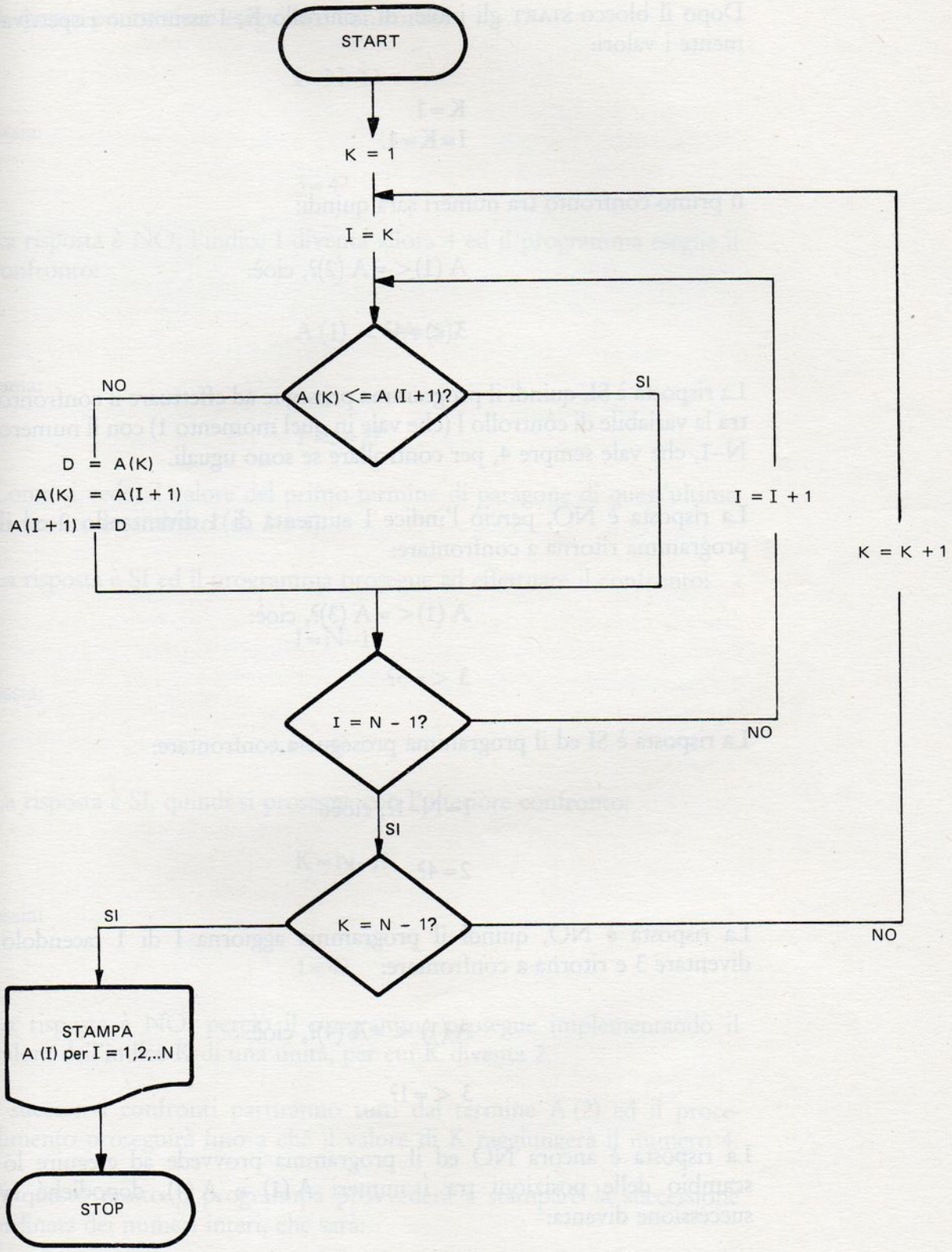



*Fig. 12 – Flow-chart dell'ordinamento di N numeri interi in ordine crescente.*

Dopo il blocco START gli indici di controllo K, I assumono rispettivamente i valori:

K=1
I=K=1.

Il primo confronto tra numeri sarà quindi:

A (1)< =A (2)?, cioè:

3 <=4?

La risposta è SI, quindi il programma prosegue ad effettuare il confronto tra la variabile di controllo I (che vale in quel momento 1) con il numero N–1, che vale sempre 4, per controllare se sono uguali.

La risposta è NO, perciò l'indice I aumenta di 1 diventando 2 ed il programma ritorna a confrontare:

A (1)< =A (3)?, cioè:

3 <=5?

La risposta è SI ed il programma prosegue a confrontare:

I=N–1?, cioè:

2=4?

La risposta è NO, quindi il programma aggiorna I di 1 facendolo diventare 3 e ritorna a confrontare:

A (1) <=A (4)?, cioè:

3 <=1?

La risposta è ancora NO ed il programma provvede ad eseguire lo scambio delle posizioni tra i numeri A (1) e A (4), dopodiché la successione diventa:

1, 4, 5, 3, 2.

A questo punto viene eseguito il confronto:

I=N-1?

ossia:

3=4?

La risposta è NO; l'indice I diventa allora 4 ed il programma esegue il confronto:

A (1) <=A (5)?

ossia:

1 <=2?

Come si vede, il valore del primo termine di paragone di quest'ultimo confronto è cambiato (da 3 a 1).

La risposta è SI ed il programma prosegue ad effettuare il confronto:

I=N-1?

ossia:

4=4?

La risposta è SI, quindi si prosegue con l'ulteriore confronto:

K=N-1?

ossia:

1=4?

La risposta è NO, perciò il programma prosegue implementando il valore dell'indice K di una unità, per cui K diventa 2.

I successivi confronti partiranno tutti dal termine A (2) ed il procedimento proseguirà fino a che il valore di K raggiungerà il numero 4.

A questo punto il programma provvederà a stampare la successione ordinata dei numeri interi, che sarà:

1, 2, 3, 4, 5.

## 7. - Linguaggi di programmazione e sistemi operativi

Il programma rappresentato in diagramma di flusso, per poter essere introdotto nel computer, deve essere formalizzato in un frasario simbolico, chiamato **linguaggio di programmazione.**

Le istruzioni operative, di diramazione, di assegnazione, ecc. vengono scritte con un particolare simbolismo, che è molto simile al frasario algebrico-matematico e la cui lettura è quindi comprensibile.

I linguaggi di programmazione sono differenziati in base alla loro maggiore o minore corrispondenza strutturale con il linguaggio macchina, che è l'espressione assoluta, non piú trasformabile, delle operazioni logico-aritmetiche del programma, scritto in rappresentazione binaria.

Nella *fig. 13* è rappresentato in forma schematica il sistema nel quale si definiscono gli ordinari programmi dei microcomputer.

In questa rappresentazione il programma occupa la posizione centrale del sistema. Occorre notare che il programmatore deve sempre svolgere una ricerca preliminare, avente per oggetto la definizione del procedimento di risoluzione del problema (algoritmo).

Il programma, scritto con particolari simboli, ossia in un determinato

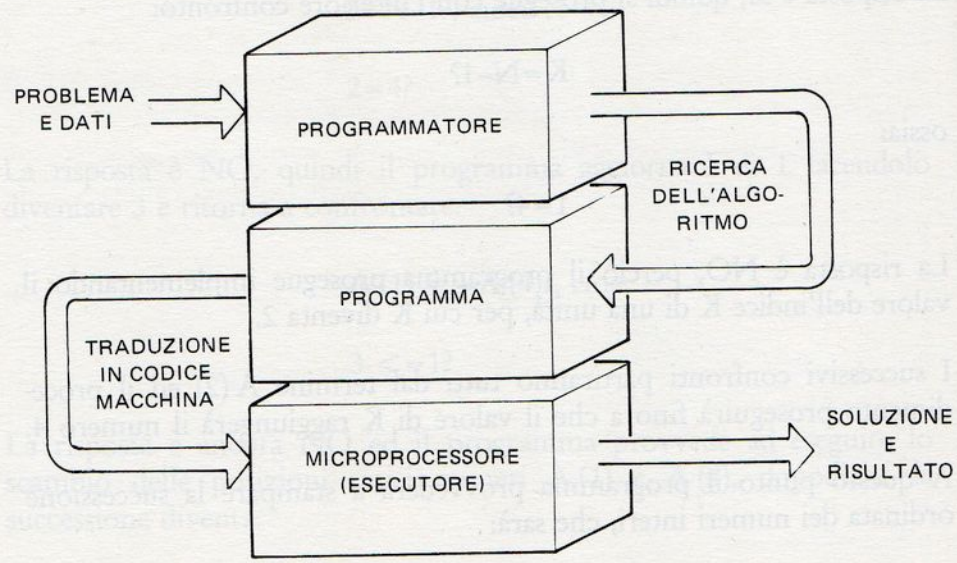



*Fig. 13 - Sistema nel quale si definiscono gli ordinari programmi dei microcomputer.*

linguaggio, viene poi tradotto in codice macchina, affinché il microcomputer possa trasformare automaticamente un certo problema ed i relativi dati nella soluzione e nei rispettivi risultati.

Come già precedentemente precisato, l'impiego delle istruzioni in linguaggio macchina presenta notevoli difficoltà per l'uomo, poiché si devono manipolare, senza commettere assolutamente errori, lunghe sequenze di cifre binarie per ogni singola istruzione.

Per rendere piú agevole il colloquio uomo-computer, sono state elaborate due categorie di linguaggi simbolici di programmazione: il **linguaggio assemblativo** ed i **linguaggi evoluti ad alto livello.**

Vediamo ora molto brevemente le caratteristiche peculiari di tali linguaggi, senza però entrare nei dettagli.

**L'assembly**

Nel linguaggio assemblativo (ASSEMBLY) la codificazione in linguaggio macchina di ogni istruzione è costituita da una abbreviazione mnemonica o simbolica, che riassume ciascuna tappa elementare del calcolo o dell'elaborazione che il computer deve eseguire.

**L'assembler**

Naturalmente per il funzionamento del computer il linguaggio assembly deve venire successivamente tradotto nel corrispondente programma in linguaggio macchina, il che viene attuato tramite un **programma assemblatore** (ASSEMBLER), contenuto nella memoria del computer.

A volte nelle pubblicazioni tecniche e nei testi scientifici il linguaggio assembly viene impropriamente chiamato assembler.

Il linguaggio assembly è ancora un linguaggio proprio della macchina, della quale riflette la struttura, per cui ogni computer ha un proprio linguaggio assemblativo.

La corrispondenza o analogia fra il linguaggio macchina ed il linguaggio assembly fa sí che essi siano considerati linguaggi **orientati** alla macchina, a differenza degli altri linguaggi di livello superiore.

I linguaggi ad alto livello consentono invece di ridurre notevolmente il lavoro di stesura di un programma.

Questi linguaggi di programmazione raggruppano diverse istruzioni in un'unica frase o espressione, per cui il programma si presenta costituito da una successione di tali frasi.

**Il compiler**

Il linguaggio di programmazione ad alto livello risulta quindi estremamente sintetico ed è tradotto in linguaggio macchina da un **programma compilatore** (COMPILER) o **interprete**, contenuto nel computer.

In questi anni sono state sviluppate centinaia di linguaggi di programmazione, anche se sono soltanto una ventina quelli utilizzati piú diffusamente.

**Linguaggi evoluti**

Fra i linguaggi evoluti della prima generazione si possono citare il FORTRAN ed il COBOL; tra quelli della seconda generazione l'ALGOL, l'APL, il PL/1, il BASIC, il PASCAL, l'ADA, ecc.

Il FORTRAN (*FORmula-TRANslation*, ossia traduttore di formule), apparso nel 1955, è un linguaggio cosiddetto algoritmico perché, in analogia con gli algoritmi matematici, è formalmente simile al simbolismo algebrico. Esso è molto usato nella risoluzione di problemi scientifici.

Il COBOL (*COmmon Business Orientated Language*) è un linguaggio simbolico di tipo discorsivo ed ha una struttura sostanzialmente diversa dal Fortran. E' utilizzato per le applicazioni gestionali ed amministrative, dove si richiede l'elaborazione di un numero piuttosto cospicuo di dati.

Con questo linguaggio, infatti, i dati (numerici ed alfanumerici) sono organizzati in strutture generiche di una certa complessità, per cui diventa abbastanza agevole accedere ad una informazione.

L'ALGOL (*ALGorithmic Orientated Language*, ossia linguaggio per algoritmi), presentato come successore del Fortran, non ebbe però la diffusione sperata; esso fu adottato principalmente dalle università e divenne la base dei linguaggi strutturati Pascal e Ada.

L'APL (*A Programming Language*) è un linguaggio creato per essere usato con calcolatori aventi grandi memorie. Esso è caratterizzato dal fatto di avere un vocabolario molto ricco; viene impiegato in campo didattico e nelle ricerche.

Il PL/1 (*Programming Language/One*) ha molte caratteristiche del Fortran ed offre alcune tra le migliori possibilità di altri linguaggi, come il Cobol e l'Algol. E' stato il primo linguaggio adattabile sia all'analisi scientifica sia agli usi aziendali; viene impiegato nei calcolatori di grandi dimensioni.

Il BASIC (*Beginner's All-purpose Symbolic Instruction Code*, ossia codice di

istruzione simbolico ad uso generale per il principante) è il linguaggio di programmazione probabilmente piú facile da imparare ad usare. E' d'uso generale ed è senz'altro il piú diffuso nel campo dei microcomputer; infatti è quello maggiormente conosciuto fra i vari linguaggi.

Il PASCAL è un linguaggio derivato dall'Algol e costituisce il primo esempio di linguaggio di programmazione strutturata. Un programma strutturato è essenzialmente costituito di parti disposte una dentro all'altra anziché affiancate ed eseguite in sequenza.

I principali vantaggi presentati dal Pascal sono: possibilità di operare su strutture di dati comunque complicate, potenza matematica paragonabile a quella dei principali linguaggi scientifici, tecnica di programmazione strutturata.

L'ADA (cosí chiamato in onore di Ada Lovelace, figlia di lord Byron ed emula di Charles Barbage) è un linguaggio derivato dal Pascal. E' anch'esso un linguaggio strutturato ed i suoi principali requisiti sono: modularietà dei programmi, sicurezza di analisi, uso efficiente del calcolatore.

A conclusione di questa breve ed incompleta panoramica sul mondo dei linguaggi di programmazione, si possono ancora citare i linguaggi specializzati ai fini di un particolare uso tecnico o scientifico, quali per esempio il LIPS (*List Processing Language*) ed il FORTH (*Fourt Generation*).

## 7.1 - Il sistema operativo

Il **sistema operativo** di un calcolatore è costituito da un insieme di programmi, già presenti nella memoria del computer, che servono alla gestione operativa dell'elaborazione; con tale termine si intende tutto il complesso delle funzioni di servizio utilizzate per dirigere e controllare le varie fasi d'elaborazione del programma che si vuole eseguire. Le fasi d'elaborazione possono essere un'immissione di dati, una stampa di risultati, una lettura di informazioni da nastri e dischi, una scrittura di informazioni su nastri e dischi, una perforazione, ecc.

**Multiprogrammazione**

Una funzione importante del sistema operativo, che però viene svolta sui grossi calcolatori, è la MULTIPROGRAMMAZIONE.

La multiprogrammazione è un modo di operare a **segmenti paralleli**, ossia i programmi presenti in memoria vengono elaborati alternativamente un po' per volta.

Prima viene elaborato un programma avente una certa priorità, poi si interrompe questo e, dopo averne conservato lo "stato", cioè il complesso delle variabili ed il punto di programma eseguito fino a quel momento, si passa ad elaborare il segmento di un altro programma, e cosí via fino ad esaurire tutti i programmi presenti in memoria.

Terminata una prima serie di elaborazioni, il computer ritorna ad eseguire il primo programma.

E' una tecnica piuttosto complessa, che consente però al sistema di elaborazione di funzionare al "meglio", cioè permette all'unità centrale di "lavorare sempre", evitando ritardi, pause ed interruzioni.

Nei piccoli sistemi di elaborazione esiste soltanto la monoprogrammazione, cioè un solo programma per volta può essere elaborato. Stanno però per essere introdotti anche per i personal computer dei sistemi operativi in grado di effettuare la multiprogrammazione di due o tre programmi.

## 8. – Generalità e struttura del BASIC

Questo linguaggio, come già precedentemente precisato, è il piú comunemente usato nei microcomputer e nei personal computer.

La sua semplicità e facilità d'apprendimento derivano dal fatto che esso possiede caratteristiche "conversazionali". In effetti, usando il linguaggio di programmazione BASIC, il programmatore ha l'impressione di conversare con il calcolatore.

A differenza di altri linguaggi, il BASIC viene interpretato; ogni istruzione (o comando) da cui è costituito il programma viene introdotta nel computer attraverso i simboli grafici, letterali e numerici della tastiera e viene interpretata come una normale espressione linguistica. Se questa espressione è stata scritta correttamente, il computer la esegue, altrimenti segnala l'errore sintattico sullo schermo, in modo che il programmatore possa correggerla.

L'interpretazione e l'esecuzione da parte del computer avvengono per ogni singola istruzione senza compiere una vera e propria analisi sintattica di tutto il programma, tramite un traduttore denominato appunto **interprete.**

E' questo un modo di operare interattivo, valido soltanto per la classe dei linguaggi interpretati BASIC.

Se da un lato ciò facilita l'accesso alla programmazione sul computer, dall'altro costituisce un onere non indifferente per quest'ultimo.

Infatti, se si desidera programmare il problema n. 3 visto precedentemente, il quale prevede l'ordinamento in modo crescente di una serie di N numeri reali, si constata subito che, al crescere di N (numero dei dati da ordinare), aumenta il tempo che il computer impiega prima ad interpretare e poi ad eseguire il gruppo di istruzioni BASIC che realizza l'ordinamento.

Invece, nei linguaggi compilati, tipo il Fortran e il Cobol, la fase di interpretazione e traduzione in linguaggio macchina binario avviene una volta soltanto, all'inizio, cioè prima dell'esecuzione. Con questi linguaggi non è infatti possibile interpretare le istruzioni a mano a mano, come si è visto per il BASIC: occorre che il traduttore (compilatore) esegua prima un'analisi sintattica completa di tutto il programma e generi il codice in linguaggio macchina passo a passo e poi lo esegua quando il programma viene avviato.

Un programma scritto in linguaggio BASIC è un insieme di istruzioni numerate.

Tali numeri, che rappresentano dei riferimenti (in inglese LABEL) sono valori interi in ordine crescente, non necessariamente in successione naturale, cioè crescenti di uno in uno.

Normalmente i numeri delle istruzioni BASIC aumentano di 10 in 10; ciò serve ad effettuare degli eventuali inserimenti senza dover ribattere le istruzioni susseguenti.

Se, per esempio, dopo aver battuto il seguente programma:

```
10 REM....
20 PRINT....
30 INPUT....
40 REM....
50 A=....
60 IF A<3....
:
```

si desidera inserire altre istruzioni tra l'istruzione 40 e l'istruzione 50,

basta contraddistinguere queste nuove istruzioni con numeri di riferimento compresi tra 4Ø e 5Ø.

Se l'istruzione non fa parte di un programma, cioè va eseguita singolarmente ed immediatamente (si dice anche "istruzione ad esecuzione immediata"), il numero di riferimento non va riportato.

Non tutte le istruzioni BASIC sono contemporaneamente ad esecuzione immediata e differita. L'istruzione **PRINT** può essere ad esecuzione immediata e ad esecuzione differita; l'istruzione **INPUT** può essere solo ad esecuzione differita.

Le istruzioni BASIC si dividono sostanzialmente in due gruppi: ad un gruppo appartengono le istruzioni vere e proprie, dette anche **enunciati**, all'altro gruppo i **comandi**.

**Le istruzioni BASIC**

Le istruzioni, alcune delle quali possono essere ad esecuzione immediata e differita, sono utilizzate quasi sempre per l'elaborazione dei programmi e la raffigurazione visiva (sullo schermo del monitor o su una stampante) dei risultati.

**I comandi BASIC**

I comandi, la maggior parte dei quali è ad esecuzione immediata, cioè viene usata all'infuori dei programmi, sono utilizzati per la gestione automatica delle unità periferiche, come la stampante, l'unità a disco, l'unità a nastro, ecc.

Di seguito sono elencati le istruzioni ed i comandi BASIC piú comunemente in uso.

*ISTRUZIONI BASIC TIPICHE*

**PRINT** Si usa per visualizzare sullo schermo i dati.

**INPUT** Serve per l'immissione tramite tastiera di dati e parole.

**LET** E' usata per assegnare valori numerici e parole alfanumeriche a variabili.

**IF... THEN...** Consente una diramazione al programma di elaborazione in base al valore di una relazione.

**FOR.. NEXT...** Consente al programma di effettuare una elaborazione ciclica.

*COMANDI BASIC TIPICI*

**NEW** Consente di cancellare o "azzerare", come spesso si dice, programmi e dati contenuti nella memoria principale del computer.

**LIST** E' usato per listare le istruzioni di un programma e visualizzarle sullo schermo.

**LOAD** Serve per memorizzare o "caricare" programmi registrati sull'unità a nastro o a disco nella memoria principale del computer.

**RUN** Consente di eseguire un programma memorizzato o caricato in memoria.

**SAVE** E' usato per registrare o "salvare", come si dice anche, un programma dalla memoria principale su una unità a disco o a nastro.

**CATALOG** Consente di elencare la lista dei programmi archiviati sull'unità a disco.

**LOCK** Provvede a "preservare" dalla scrittura programmi registrati su disco.

**MON, NOMON** Permette di visualizzare le istruzioni sullo schermo nel momento in cui queste vengono eseguite.

I comandi BASIC hanno caratteristiche operative, mentre le istruzioni riguardano maggiormente la logica dell'elaborazione.

Le istruzioni fanno parte del linguaggio vero e proprio di programmazione BASIC, mentre i comandi sono inseriti nel sistema operativo che, come si è visto, costituisce il complesso di programmi già presenti nella memoria del computer, avente funzioni di gestione organizzativa ed operativa dell'elaborazione.

Si incomincia a parlare però di vero e proprio sistema operativo nel caso di personal computer dotati di unità a disco e con piú di 16 Kbyte di memoria principale. Nei microsistemi come il VIC-20 (con appena 5 Kbyte di memoria) il sistema operativo non esiste e praticamente i pochi comandi esistenti sono inseriti nel linguaggio BASIC.

Nei personal computer la fase che consente l'aggiunta o l'implementazione dei comandi operativi alle istruzioni del linguaggio si chiama BOOT (dall'inglese "calzare", cioè completare, aggiungere) ed avviene normalmente in fase di accensione del computer.

Nei grossi sistemi di elaborazione il BOOT viene effettuato attraverso una procedura di inizializzazione piuttosto laboriosa e complessa.

Il BOOT di un sistema di elaborazione costituisce la fase preliminare per poter svolgere qualsiasi elaborazione; con esso cioè il computer è pronto a ricevere qualsiasi istruzione da parte dell'operatore.

Nei personal computer la segnalazione che il sistema è "pronto" per elaborare è visualizzata sul monitor con frasi significative e con l'apparizione del cosiddetto carattere "pronto" e del cursore; quest'ultimo può essere costituito da un quadratino lampeggiante, oppure da un carattere speciale.

Ad esempio, nel sistema APPLE II, il carattere pronto viene visualizzato con la parentesi quadra di chiusura seguita dal cursore (quadrato lampeggiante); nel sistema VIC-20 il carattere pronto è costituito dalla parola inglese READY che significa appunto "PRONTO".

Ritornando alle istruzioni del linguaggio BASIC, alcune di esse possono essere veri e propri comandi, inseriti in un programma per svolgere funzioni operative, come la registrazione o la lettura di unità di memoria periferiche o la stampa di risultati su una stampante o la perforazione di dati su una perforatrice, ecc.

Per esempio l'istruzione **INPUT**, se scritta con il carattere speciale # (cancelletto), come indicato di seguito, consente l'immissione di dati dall'unità a disco o a nastro in memoria:

```
10 INPUT # 1, "DATI"
```

Questa istruzione permette l'immissione del file di nome "DATI", contenente dei numeri, dall'unità a nastro (unità periferica a cui è stato assegnato il numero 1) nella memoria principale.

In altri sistemi personal computer è consentito trasferire il controllo del programma dalla tastiera del computer ad un terminale stampante, per cui la visualizzazione sia del programma sia dei risultati dell'elaborazione può avvenire su quest'ultima periferica; ciò è consentito scrivendo particolari istruzioni di trasferimento, come per esempio le seguenti

(valide per il computer Commodore):

```
OPEN 1,4
CMD 1
LIST
```

Il comando **OPEN** 1,4 apre il dispositivo n° 4, che è l'unità periferica 4 (per esempio, un terminale dotato di stampante); il comando **CMD** 1 trasferisce il file dati 1 alla stampante; il comando **LIST** stampa il programma.

Altri comandi operativi, quali: **CATALOG, DELETE, SAVE, LIST, LOAD, EXEC,** ecc. possono essere inseriti nella sequenza di un programma se vengono caratterizzati da altri simboli speciali.

In questa lezione si è tracciato un primo prospetto del linguaggio BASIC, sufficiente per affrontare i primi elementi di programmazione su un personal computer.

* * * * * * * *

## In seguito...

**Nella lezione successiva saranno ripresi in modo piú organico i concetti e le nozioni sin qui enunciati, completando ed arricchendo con altri particolari la descrizione delle caratteristiche del linguaggio BASIC.**

## Da memorizzare

- Il programma di un computer è costituito da un insieme ordinato di istruzioni, scritte secondo un frasario simbolico e in base a precise regole sintattiche.
- Il programma viene immesso nel calcolatore mediante un lettore di schede (elaborazione batch), oppure gradualmente tramite organi di input: tastiere, telescriventi (elaborazione time-sharing).
- Il programma scritto in **linguaggio simbolico** viene tradotto nell'interno del computer in **linguaggio macchina**, cioè in un sistema di rappresentazione binaria, formato dai simboli 1 e 0, il solo che può essere elaborato dal calcolatore.
- Le istruzioni di un programma devono essere scritte con un frasario di tipo algebrico ed una sintassi precisa; inoltre, l'**algoritmo di risoluzione** deve convergere alla soluzione del problema con un numero finito di passi.
- Il diagramma di flusso (**flow-chart**) rappresenta l'espressione visiva dell'algoritmo, cioè la serie completa delle istruzioni necessarie per risolvere, passo a passo, un dato problema con il computer.
- I problemi risolvibili con il calcolatore sono di natura digitale o riconducibili ad essa.
- 1 Kbyte equivale a 1024 byte.
- Il **loop** (ciclo) è una successione di operazioni che devono essere ripetutamente eseguite dal calcolatore.
- La memoria **PROM** è una memoria ROM che può essere programmata una sola volta dall'utente prima di essere utilizzata.
- La memoria **EPROM** è una memoria ROM che può essere cancellata e nuovamente riscritta ogni volta che l'utente lo desidera.
- Vi sono due categorie di linguaggi simbolici: il linguaggio assemblatore (**ASSEMBLY**) ed i linguaggi evoluti ad alto livello.
- Nell'interno del computer il linguaggio assembly viene tradotto in linguaggio macchina da un apposito programma, denominato **ASSEMBLER**.
- Nell'interno del calcolatore i linguaggi ad alto livello vengono tradotti in linguaggio macchina da un programma compilatore (**COMPILER**) o da un programma **interprete**.
- Il sistema operativo di un calcolatore è costituito da un insieme di programmi, già presenti nella memoria principale, i quali servono alla gestione operativa dell'elaborazione.
- Il programma compilatore esegue un'analisi sintattica **completa di tutto il programma** prima di tradurlo, passo a passo, e di eseguirlo.
- Il programma interprete traduce **una per volta le istruzioni del programma**, senza eseguire una vera e propria analisi sintattica del programma stesso.

- Un programma scritto in linguaggio BASIC è un insieme di istruzioni numerate.
- Le istruzioni del linguaggio BASIC si dividono in due gruppi: istruzioni vere e proprie, dette anche **enunciati**, e **comandi**.

Scuola Radio Elettra
Torino

# Pratica del BASIC

## Lezione 3

1 — L'istruzione **INPUT**
2 — Le istruzioni **GOTO...** e **IF... THEN...**
3 — L'istruzione **REM**
4 — Esercizi di programmazione

## Che cosa farà

In questa lezione apprenderà l'uso di alcune altre istruzioni BASIC, come la **INPUT** (per l'introduzione di dati da tastiera) e le istruzioni di diramazione **GOTO**... e **IF**... **THEN**....

In particolare, come potrà verificare attraverso gli appositi esercizi di programmazione previsti, le istruzioni di diramazione sono alla base di qualsiasi decisione logica che deve compiere il computer.

## 1. – L'istruzione **INPUT**

L'istruzione **INPUT** serve per l'introduzione dei dati, durante l'esecuzione di un programma, dall'unità d'ingresso principale del computer, cioè dalla tastiera.

Il formato standard di questa istruzione è il seguente:

numero riga **INPUT** nome variabile

Il numero riga è il numero o label di riferimento (etichetta) dell'istruzione **INPUT** nell'ambito del programma.

**L'istruzione INPUT è ad esecuzione differita, quindi riporta sempre il numero di una riga.**

A questo riguardo occorre tenere presente che un programma è sempre costituito da un numero di righe scritte, dette anche **linee**, ognuna delle quali contiene una o piú istruzioni per il calcolatore.

Con questa istruzione viene anche indicato il nome di una variabile, nome che contraddistingue la zona di memoria (chiamata in precedenza "cassetto" di memoria) dove va memorizzato il valore del dato che si introduce con la tastiera.

Per esempio, l'istruzione:

```
80 INPUT A
```

consente di introdurre da tastiera un dato numerico nella zona di memoria chiamata A.

A che cosa serve denominare od etichettare le zone di memoria dove vanno a finire i dati da elaborare?

Serve a richiamarle, quando è necessario, nel proseguimento del programma di elaborazione; se per esempio nel "cassetto" di nome A è stata introdotta la costante $\pi$ (pi greco=3,14159) e questa costante viene menzionata nella successiva istruzione di programma che calcola l'area di un cerchio:

```
100 S=A*R↑2
```

il computer, nel momento in cui deve calcolare la suddetta espressione, utilizzerà la costante 3.14159 al posto della lettera A, cioè la preleverà dal "cassetto" di memoria chiamato A.

L'istruzione:

```
80 INPUT A
```

verrà eseguita successivamente; ricordi che l'esecuzione di un programma contenente un insieme di istruzioni numerate in ordine crescente avviene dopo che esso è stato introdotto nella memoria del computer e si effettua battendo sulla tastiera il comando BASIC:

```
RUN
```

seguito sempre dal tasto **RETURN**.

Quando l'elaborazione arriva all'istruzione:

```
80 INPUT A
```

si vede apparire sul margine sinistro dello schermo il punto interrogativo:

A questo punto l'operatore batterà con la tastiera il numero richiesto (cioè 3.14159), seguito sempre dal tasto **RETURN** e l'elaborazione proseguirà con l'istruzione successiva del programma.

Si è cosí certi che il computer ha incamerato il numero 3.14159 nel "cassetto" di memoria chiamato A.

A proposito dei nomi che si possono attribuire ai dati numerici, occorre precisare che questi nomi possono essere una qualsiasi combinazione di coppie di lettere e cifre decimali e che ogni nome riporta nella parte destra un particolare carattere, in base al tipo di variabile che rappresenta. Inoltre il nome inizia sempre con una lettera dell'alfabeto.

***Esempi di nomi di variabili contenenti numeri reali:***

AA, A1, B4, ZX, X9, SØ, S9, LL, M7, ecc.

A(I), RI(I,J), PH (I,J,K), ecc.

***Esempi di nomi di variabili contenenti soltanto numeri interi:***

AA%, AB%, C%, C1%, HP%, S%, Z%,

I%, L% (I,J), MØ%(I), NO%(I,K), ecc.

***Esempi di nomi di variabili contenenti qualsiasi carattere alfabetico e numerico (ossia alfanumerico):***

A$, S$, ST$, S9$, HP$, KG$, GG$, S9$(I,J), SP$(I), ecc.

Come si vede, i numeri interi sono rappresentati con nomi formati al massimo da due caratteri alfanumerici, terminanti sempre con il carattere %; le parole o stringhe (sequenze) alfanumeriche, come sono anche chiamate, vanno rappresentate con nomi costituiti al massimo da due caratteri alfanumerici, terminanti sempre con il carattere $, mentre i nomi di variabili reali (numeri interi e decimali) non hanno bisogno di alcun carattere speciale. Non si possono usare, come nomi di variabili le seguenti parole chiave del linguaggio BASIC: **IF, ON, OR, TO, GO, FN, TI, ST.**

Ritornando all'istruzione prima citata:

```
80 INPUT A,
```

se al momento dell'immissione si fosse battuta la parola alfanumerica PI-GRECO, il computer avrebbe segnalato ERRORE, perché l'istruzione in questione richiede soltanto valori numerici (come il numero 3.14159).

Se si volesse introdurre veramente una stringa alfanumerica (per esempio, nome e cognome di una persona), bisognerebbe scrivere l'istruzione **INPUT** in questo modo:

```
80 INPUT NC$,
```

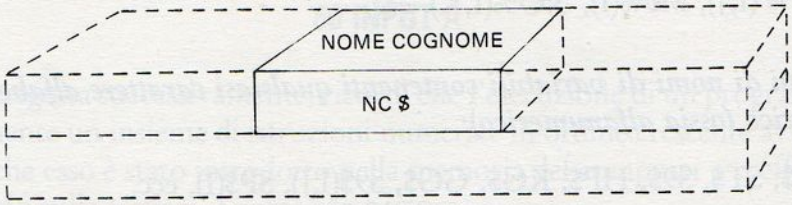
dove con NC$ viene indicato il nome della variabile stringa, il cui valore dovrà essere immesso successivamente (all'apparire del carattere ? sullo schermo).

Quando dallo schermo viene richiesto il valore stringa, si deve battere quest'ultima nel seguente modo:

"NOME COGNOME"

cioè racchiusa tra virgolette.

Dopo aver premuto il tasto **RETURN** , il computer memorizzerà la stringa: NOME COGNOME nel cassetto di memoria di nome NC$, come sotto indicato




Abbiamo fornito cosí le nozioni preliminari per l'utilizzo dell'istruzione **INPUT**; successivamente analizzeremo piú nei dettagli alcune altre particolarità di questa istruzione.

Si noti ancora, a proposito dei nomi di variabili, che in alcuni sistemi personal computer sono ammessi nomi fino ad un massimo di sei caratteri alfanumerici (per esempio, nomi come: AREA, SOMMA, TOTALE, ecc.).

## 2. – Le istruzioni GO TO... e IF... THEN...

Le istruzioni **GO TO...** e **IF... THEN...** sono **istruzioni di diramazione:** il programma di elaborazione, cioè, "si dirama" o "devia" verso l'istruzione il cui numero di riferimento è indicato nel tratto punteggiato.

Per esempio, se ad un certo punto dell'elaborazione si vuole saltare incondizionatamente all'istruzione con numero di riferimento 40, si scriverà:

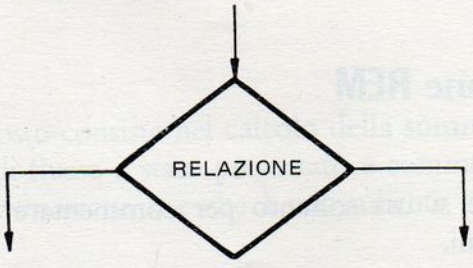
```
numero riga   GO TO  40
```

L'elaborazione, quando passa per questa istruzione, esegue un "salto" all'istruzione numero 40, cioè continua dall'istruzione numero 40.

L'istruzione **GO TO...** (che significa "vai a") è un salto senza condizioni, mentre l'istruzione **IF... THEN...** è un salto condizionato dal valore della relazione logica riportata nel tratto punteggiato dopo la parola riservata IF.

Se il valore della relazione è "vero", l'elaborazione "si dirama" o "devia" verso l'istruzione che ha il numero di riferimento indicato dopo la parola riservata **THEN**; se, al contrario, tale valore risulta "falso", l'elaborazione prosegue per l'istruzione successiva all'istruzione **IF... THEN....**

L'istruzione **IF... THEN...** è ad esecuzione differita e corrisponde al seguente simbolo grafico del flow-chart.

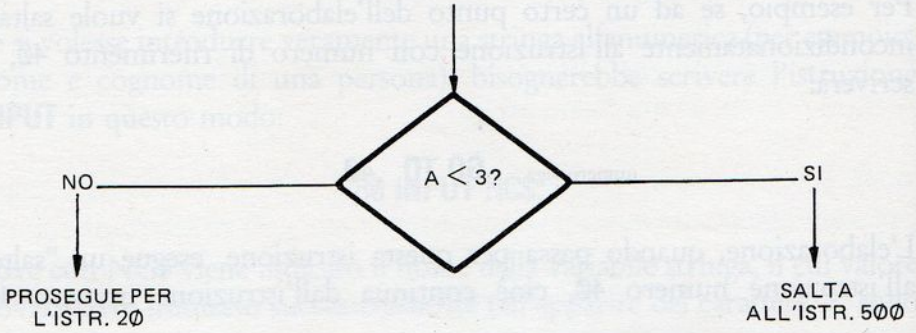



Vediamo ora un esempio di tale istruzione e precisamente quella riportata di seguito:

```
10 IF A<3 THEN 500
20.....
```

Essa significa che, **se** (infatti, IF in inglese vuol dire se) il valore della variabile di nome A risulta minore di 3, **allora** (in inglese, THEN) l'elaborazione prosegue per l'istruzione con numero di riferimento 5ØØ; se invece tale condizione non risulta verificata, ossia se A risulta "non minore" di 3 (maggiore o uguale a 3), il programma prosegue per l'istruzione 2Ø.

Quanto ora asserito è rappresentato nel flow-chart nel seguente modo.




L'istruzione **IF**, cosí come è stata scritta, corrisponde quindi ad una **alternativa sempiice.** Con questo termine si indica una condizione logica dell'elaborazione, secondo la quale il flusso di programma o prosegue da una parte o prosegue dall'altra.

In seguito, studieremo e realizzeremo alternative piú complesse, quali le **alternative multiple**, ecc., che consentono appunto diramazioni multiple.

Inoltre analizzeremo le cosiddette istruzioni **IF** nidificate, cioè sequenze logiche di alternative semplici, e le istruzioni **IF** con relazioni logiche multiple di tipo simile alle relazioni binarie dell'algebra booleana, come si vedrà in seguito.

## 3. – L'istruzione REM

Questa istruzione si usa soltanto per commentare delle istruzioni o gruppi di istruzioni.

L'istruzione **REM** (da REMARK, termine inglese che significa "commento") non ha alcun valore operativo; serve semplicemente per funzioni descrittive e di commento al programma stesso, quindi per fornire informazioni, a chi legge la lista di un programma, su che cosa si intende fare in quel punto.

Come tutte le istruzioni BASIC ad esecuzione differita, anche questa istruzione è dotata di numero di riferimento ed ha come oggetto di commento visualizzato sullo schermo qualsiasi espressione alfanumerica, la quale **non va racchiusa tra virgolette.**

L'istruzione **REM** si può inserire in qualsiasi punto del programma. Per esempio, il programma visto all'inizio della lezione *Analisi e Programmazione 3*, che calcola la circonferenza di un cerchio, se programmato con il linguaggio BASIC può essere cosí "commentato":

```
10 REM ************************************************
20 REM ***PROGRAMMA CALCOLO CIRCONFERENZA CERCHIO ***
27 REM
28 REM ***          FASE 1:INTRODUZIONE RAGGIO       ***

↓ (successive istruzioni)
```

Nell'esempio precedentemente considerato si è utilizzato un commento per ogni riga visualizzata sullo schermo.

L'espressione alfanumerica riportata dopo la parola riservata **REM** può riempire al massimo due righe dello schermo TV (80 caratteri) nel computer C64 e quattro righe (160 caratteri) nel computer C128.

## 4. – Esercizi di programmazione

Avvalendosi delle conoscenze acquisite, realizzerà ora semplici programmi sul microcomputer.

*1° ESERCIZIO*

L'esercizio proposto consiste nel **calcolo della somma di N numeri,** il cui diagramma di flusso è stato presentato e commentato nella lezione *Analisi e Programmazione 3,* e che è nuovamente riportato nella *fig. 1.*

In primo luogo si scriveranno le appropriate istruzioni in linguaggio BASIC per ognuno dei sette blocchi del flow-chart in questione, poi si procederà all'esecuzione vera e propria del programma.

Naturalmente non dovranno essere necessariamente sette le istruzioni BASIC. Come constaterà, qualche istruzione in piú di commento va inserita

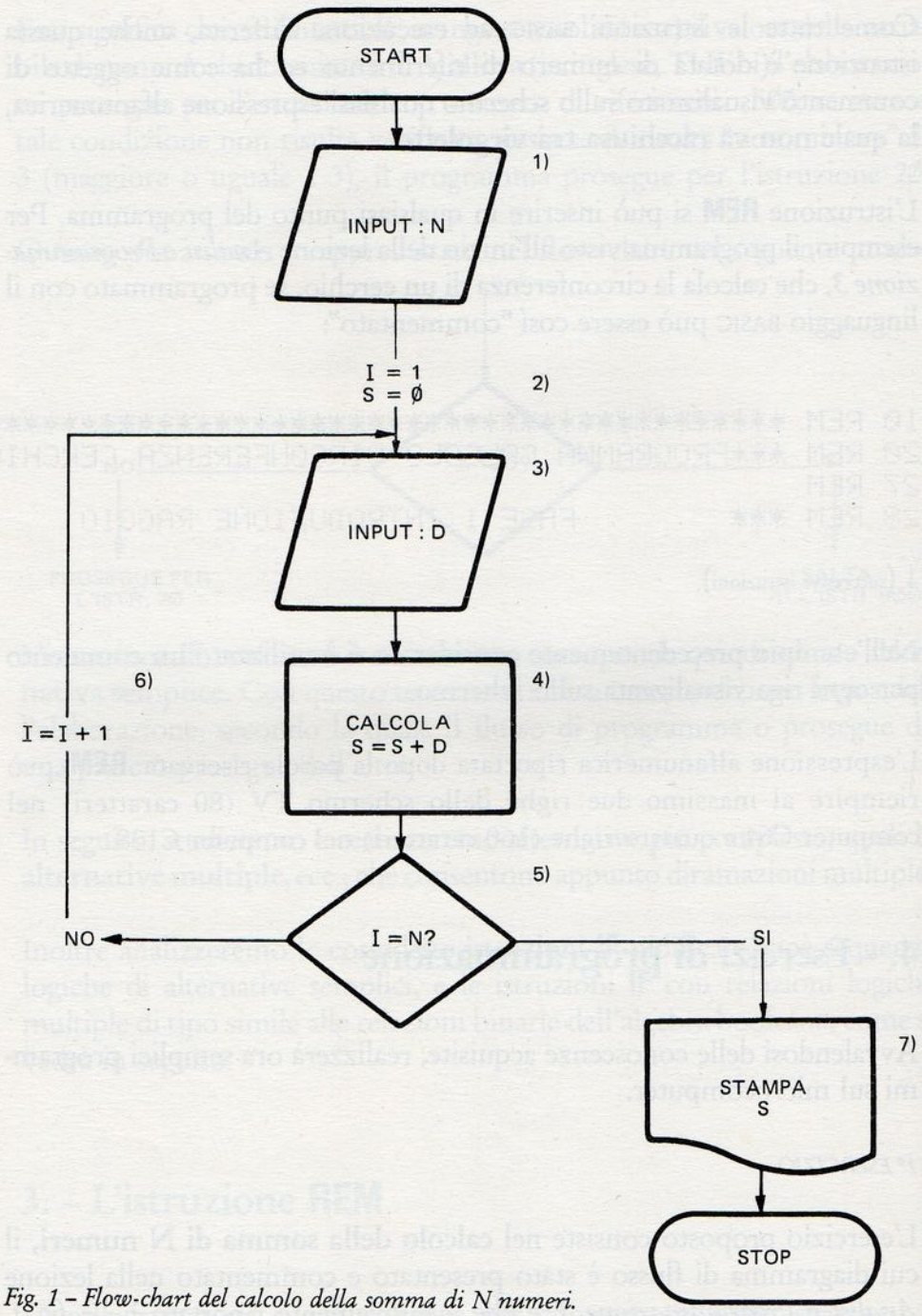



*Fig. 1 – Flow-chart del calcolo della somma di N numeri.*

al momento opportuno: per esempio, all'inizio ed alla fine di un programma, oppure in punti particolari (cosiddetti critici) che risultino sempre evidenziati al programmatore.

Ci sono poi istruzioni che non possono essere, per cosí dire, fissate con un numero di riferimento come il "ciclo" di programma, il "salto" incondizionato, ecc.

Per evidenziare l'inizio (START) del programma, si possono scrivere le seguenti istruzioni di commento:

Questo gruppo di istruzioni serve ad evidenziare, in fase di immissione del programma, la natura del problema che si sta programmando.

Si fa quindi precedere il blocco 1) del diagramma della *fig. 1* da un'indicazione, oppure messaggio, che dica all'operatore quale tipo di numero deve introdurre da tastiera.

A questo scopo si può eseguire un'istruzione che visualizzi sullo schermo TV il messaggio, ad esempio scrivere la seguente istruzione:

```
40 PRINT "INTRODURRE N (NUMERO DATI DA SOMMARE)"
```

Con questa istruzione infatti il computer, al momento dell'esecuzione, si ricorderà che l'esecuzione stessa avviene successivamente all'immissione del programma completo in memoria e provvederà a visualizzare sullo schermo il messaggio seguente:

```
INTRODURRE N (NUMERO DATI DA SOMMARE)
```

In tal modo l'operatore capirà che dovrà introdurre, quando richiesto, un numero intero N.

Come visto, l'istruzione **PRINT** ora considerata contiene un messaggio racchiuso fra virgolette, comprendente meno di quaranta caratteri, che verrà quindi visualizzato in una sola linea dello schermo.

Quando invece il messaggio racchiuso fra le virgolette è composto da piú di quaranta caratteri, compresi gli spazi, bisogna tenerne conto quando viene battuto, in modo che esso, quando verrà eseguito, sia leggibile come un'espressione letterale. Ciò significa che occorre, usando la barra spaziatrice, suddividere correttamente la parola a fine riga.

Si deve, infine, tenere presente che il testo contenuto nell'istruzione **PRINT** non influisce sul resto del programma, in quanto il messaggio racchiuso fra le virgolette non modifica l'elaborazione.

Dopo questa istruzione si può scrivere l'istruzione di introduzione:

```
50 INPUT N
```

Per programmare il blocco 2) del diagramma della *fig. 1*, occorre assegnare due valori numerici ad altrettante variabili (il valore 1 alla variabile di nome I ed il valore Ø alla variabile di nome S), cosicché si devono eseguire due istruzioni di assegnazione:

```
60 LET I=1
70 LET S=Ø
```

le quali, come precisato nella precedente lezione, si possono anche scrivere piú semplicemente:

```
60 I=1
70 S=Ø
```

Riguardo al blocco 3), si tratta di introdurre un dato numerico (uno dei tanti da sommare); quindi si deve scrivere l'istruzione di introduzione:

```
80 INPUT D
```

dove D è il nome della variabile in cui va memorizzato il dato numerico che si introduce.

Quando questa istruzione verrà eseguita, il computer, come succede di solito per le istruzioni **INPUT**, visualizzerà un punto interrogativo sullo schermo per segnalare all'operatore che deve introdurre un dato, seguito sempre dal tasto **RETURN** .

Nel blocco 4) viene eseguita la sommatoria dei dati e l'istruzione BASIC appropriata a questa funzione è sempre l'istruzione di assegnazione **LET**, che si scrive semplicemente:

```
90 S=S+D
```

Con questa istruzione il valore di D (dato numerico) si accumulerà via via che viene introdotto tramite la tastiera nel deposito di memoria S.

Nel blocco 5) il computer (cioè l'elaborazione) è di fronte ad un'alternativa semplice: proseguire per il blocco 7) se il valore di I è arrivato a N, oppure proseguire per il blocco 6) se tale valore non è ancora arrivato a N.

L'istruzione appropriata a questo scopo, ossia a risolvere quest'alternativa, è l'istruzione **IF... THEN...** che segue, e che è stata analizzata in questa lezione:

```
100 IF I=N THEN 130
110
120
130
```

Nell'istruzione 1ØØ, se I=N, l'elaborazione prosegue per l'istruzione con numero di riferimento 13Ø; altrimenti viene eseguito il blocco 6), cioè viene aumentato di una unità il valore precedente dell'indice I, con la seguente istruzione:

```
110 I=I+1
```

Successivamente, come indicato sul flow-chart, l'elaborazione ritorna ("cicla") all'istruzione 8Ø, che eseguiva il blocco 3), cioè viene effettuato il salto incondizionato:

```
120 GO TO 80
```

Dopo l'istruzione 12Ø si può scrivere l'istruzione che realizza il blocco 7), cioè la stampa su schermo del valore che rappresenta il risultato della somma e che è memorizzato nella variabile di nome S:

```
130 PRINT "IL RISULTATO DELLA SOMMA E': "; S
```

Il punto e virgola (;) posto prima della lettera S viene usato per collegare senza spaziature due espressioni alfanumeriche (in questo caso, un'indicazione letterale ed un numero).

Scrivendo l'istruzione **PRINT** nel modo ora visto, il valore S apparirà sullo schermo preceduto dall'indicazione:

```
IL RISULTATO DELLA SOMMA E':
```

A questo punto il programma viene arrestato con l'apposita istruzione di fine elaborazione:

```
140 END
```

```
10 REM*****************************
20 REM*CALCOLO SOMMA DI N NUMERI*
30 REM*****************************
40 PRINT"INTRODURRE N(NUMERO DATI DA SOMMARE)"
50 INPUT N
60 I=1
70 S=0
80 INPUT D
90 S=S+D
100 IF I=N THEN 130
110 I=I+1
120 GOTO 80
130 PRINT"IL RISULTATO DELLA SOMMA E':";S
140 END
```

*Fig. 2 – Listato completo del programma che calcola la somma di N numeri.*

Le istruzioni ora viste e commentate possono essere riportate in un elenco (listato del programma), come si vede nella *fig. 2*.

Esegua quindi il programma in questione con il Suo calcolatore, **tenendo ben presenti le seguenti avvertenze**, già precedentemente citate.

a) Proceda con la massima attenzione, e soprattutto **senza fretta**, eseguendo esattamente quanto indicato in ogni singola istruzione.

b) Prema sempre il tasto **RETURN** al termine di ogni istruzione.

c) Ricordi che per raffigurare i previsti spazi fra le parole ed i numeri occorre premere la barra spaziatrice, sistemata nella parte inferiore della tastiera.

d) Nel caso in cui battesse un carattere errato, potrà cancellarlo con il tasto **INST | DEL**, il quale fa tornare indietro il cursore di una posizione ad ogni battitura, facendo cosí scomparire il carattere che si trova immediatamente alla sua sinistra.

e) Quando deve introdurre un carattere scritto piú in alto sulla parte superiore di un tasto (per esempio, il carattere " sul tasto **" | 2**), ricordi che è necessario premere il tasto **SHIFT** e, mantenendolo premuto, abbassare il tasto interessato.

f) Nell'eventualità che in un'istruzione già immessa si accorgesse di aver sbagliato un carattere, potrà ribattere l'istruzione stessa dall'inizio,

partendo dallo stesso numero di riferimento. Ad esempio, dopo avere immesso con il tasto RETURN l'istruzione 70 S=O con la lettera "O" dell'alfabeto, anziché con la cifra Ø (zero), potrà correggere l'istruzione ribattendola esattamente: 70 S=Ø.

Il programma, una volta battuto e corretti gli eventuali errori, può essere listato sullo schermo.

Batta quindi il comando BASIC **LIST** e prema il tasto RETURN, come ha già fatto nella precedente lezione a proposito del programma "PG1".

Sullo schermo apparirà l'intero listato del programma battuto.

Ricontrolli nuovamente il programma in base a quanto riportato nella *fig. 2* e, se riscontra errori, ribatta dall'inizio con lo stesso numero di riferimento l'istruzione errata.

In tal modo l'istruzione corretta si sostituisce alla precedente e sarà la sola ad essere eseguita dal computer.

Se invece il listato che appare sullo schermo corrisponde perfettamente a quello della *fig. 2*, può procedere all'esecuzione del programma.

Batta quindi il comando BASIC:

**RUN**

e prema il tasto RETURN.

Il programma va in esecuzione e sullo schermo apparirà la seguente scritta:

```
INTRODURRE N (NUMERO DATI DA SOMMARE)
?
```

Per collaudare il funzionamento del programma, possiamo ad esempio eseguire la somma dei seguenti cinque numeri:

2; 3.5; 1.Ø7; 5; 7.

Prema quindi il tasto del numero 5 (numeri da sommare) e poi il tasto **RETURN**.

Sullo schermo apparirà la scritta:

```
? 5
?
```

Batta ora sulla tastiera i cinque numeri da sommare, uno di seguito all'altro, **premendo dopo ciascun numero il tasto RETURN**. Sullo schermo verrà riprodotto quanto segue:

```
? 2
? 3.5
? 1.Ø7
? 5
? 7
```

Premendo il tasto **RETURN** dopo avere scritto l'ultimo dei cinque numeri da sommare, ossia il numero 7, sullo schermo apparirà la seguente scritta:

```
IL RISULTATO DELLA SOMMA E': 18.57
READY
```

Ottenuto il risultato indicato, può essere certo che il programma "funziona" correttamente.

Per eseguire la somma di altri N numeri **deve battere nuovamente il comando RUN e premere il tasto RETURN**.

Nell'eventualità che, durante il collaudo del programma dovesse erroneamente battere, al posto del numero da sommare, una lettera dell'alfabeto, sullo schermo del televisore apparirà la scritta:

```
? REDO FROM START
```

*(rifare dall'inizio)*

Questa scritta indica appunto che è stato introdotto un carattere alfanumerico in una istruzione **INPUT**, la quale prevede solo dati numerici.

In tal caso, per proseguire nel programma, è sufficiente ribattere correttamente il numero da sommare.

Se qualche maldestra operazione porta il computer a funzionare in condizioni che Lei non può ancora controllare, occorre spegnere il computer, attendere 5 ÷ 6 secondi, riaccenderlo e ricominciare da capo l'esecuzione del programma. Se Lei dispone del computer C128 è invece sufficiente premere il tasto RESET.

Il programma ora eseguito può anche essere utilizzato per sommare numeri relativi (cioè numeri positivi e negativi).

Per fare ciò è sufficiente introdurre ogni numero negativo con il segno – (meno) davanti; per i numeri positivi non occorre anteporre ad essi il segno +.

In matematica la somma ora vista viene definita somma algebrica di N numeri.

Se il programma della somma di N numeri, come si è verificato, funziona correttamente, può registrare tale programma con il nome "PG2" sul lato B della cassetta a nastro, dopo il programma "PG1" registrato nella precedente lezione.

Per eseguire questa operazione è necessario posizionare la testina di lettura/scrittura del registratore a nastro subito dopo il programma "PG1" già registrato.

- Disponga quindi la cassetta nell'apposito vano del registratore, con il lato B rivolto verso l'alto, e riavvolga completamente il nastro premendo il tasto REWIND; quindi, prema il tasto STOP.

- Batta il comando BASIC:

**VERIFY** "PG1"

e prema il tasto **RETURN** ; sullo schermo apparirà la scritta:

PRESS PLAY ON TAPE

- Prema quindi il tasto PLAY del registratore: lo schermo si cancellerà e dopo pochi secondi apparirà il seguente messaggio:

```
OK
SEARCHING FOR PG1
FOUND PG1
```

Dopo di ciò lo schermo si cancellerà nuovamente e dopo altri pochi secondi il precedente messaggio si completerà con le scritte:

```
VERIFYING
?  VERIFY ERROR
READY
```

- Prema il tasto STOP del registratore. Con il comando ora eseguito si è posizionato il nastro magnetico nel punto giusto per un'altra registrazione, in quanto, eseguendo il comando **VERIFY** "PG1", viene fatto un raffronto tra il programma "PG1" e quello che è attualmente presente nella memoria principale del computer (somma di N numeri), e da tale confronto risulta il messaggio d'errore:

```
?  VERIFY ERROR
```

Lo scopo comunque è stato raggiunto (cioè la testina di lettura/scrittura del registratore è stata posizionata dopo il programma "PG1") e quindi a questo punto è possibile effettuare l'archiviazione del programma "somma di N numeri" con la denominazione "PG2".

- Batta quindi il comando BASIC:

```
SAVE "PG2"
```

e prema **RETURN** ; sullo schermo del televisore deve apparire il previsto messaggio:

```
PRESS RECORD & PLAY ON TAPE
```

- Prema entrambi i tasti RECORD e PLAY del registratore: lo schermo si cancellerà e dopo pochi secondi vedrà apparire le scritte:

```
OK
SAVING PG2
READY
```

- Prema il tasto STOP del registratore.

L'operazione di salvataggio del programma è cosí completata.

A questo punto è opportuno controllare il programma, per accertarsi che sia stato correttamente registrato.

- Riavvolga pertanto dall'inizio il nastro magnetico, premendo il tasto REWIND del registratore: ad avvolgimento avvenuto, prema il tasto STOP.
- Batta il comando BASIC:

**VERIFY** "PG2"

quindi prema il tasto **RETURN** ; sullo schermo comparirà la consueta scritta:

PRESS PLAY ON TAPE

Se, dopo avere premuto il tasto PLAY del registratore, tutto procede regolarmente, sullo schermo appariranno le seguenti scritte, intervallate da brevi periodi in cui lo schermo viene cancellato:

```
OK
SEARCHING FOR PG2
FOUND PG1
FOUND PG2
VERIFYING
OK
READY
```

- Al termine dell'operazione di verifica prema il tasto STOP del registratore.

Come può rilevare dal messaggio ora visualizzato sullo schermo, il computer elenca i due programmi per fare sapere che essi sono presenti sul nastro magnetico.

Le scritte FOUND PG2 e VERIFYING significano che il programma cercato è stato trovato e che è in corso la verifica di quello che è presente nella memoria principale del computer.

L'esito positivo di tale confronto è visualizzato con le parole OK e READY.

Nel caso invece dovesse comparire il messaggio:

? VERIFY ERROR

significa che la registrazione non è avvenuta correttamente e quindi occorre ripetere da capo le operazioni di registrazione del programma "PG2".

Per avere un'ulteriore conferma che il programma "PG2" è effettivamente registrato sul nastro magnetico, batta il comando BASIC **NEW** e prema il tasto **RETURN** .

In questo modo il programma presente nella memoria centrale del computer (ossia "PG2") viene cancellato.

Dopo avere riavvolto completamente il nastro magnetico con il tasto REWIND, carichi (ossia immetta) nuovamente il programma "PG2", presente sul nastro stesso, nella memoria principale del computer, battendo il comando BASIC **LOAD** "PG2" e premendo il tasto **RETURN** .

A caricamento avvenuto, può eseguire il programma battendo, come di consueto, il comando BASIC **RUN** poi il tasto **RETURN** .

Invece, per caricare nella memoria principale del computer il programma "PG1", anch'esso presente sul nastro magnetico, dovrà prima riavvolgere completamente il nastro stesso e quindi battere il comando BASIC:

**LOAD** "PG1"

**Durante l'operazione di caricamento dei programmi dal registratore alla memoria principale del calcolatore, faccia sempre bene attenzione a non premere il tasto RECORD, poiché, premendo tale tasto, il nastro magnetico viene cancellato.**

Come certamente avrà constatato, l'unità a nastro costituisce un'apparecchiatura molto utile per realizzare un archivio di programmi, ma, come per tutti gli archivi, occorre ricordare il contenuto e la posizione sul nastro magnetico di ogni programma.

Al fine di evitare inutili e laboriose ricerche, è molto utile disporre di un elenco aggiornato dei programmi registrati su nastro magnetico, ognuno contraddistinto da una propria sigla.

Tale elenco, scritto su un normale foglio di carta, dovrebbe riportare le informazioni indicate nel prospetto che segue.

| POSIZIONE SUL NASTRO | NOME DEL PROGRAMMA | NUMERO APPROSSIMATO DEL CONTANASTRO | DESCRIZIONE DEL CONTENUTO |
|---|---|---|---|
| 1ª | PG1 | 6 | Esempio di visualizzazione di una scritta sullo schermo. |
| 2ª | PG2 | 13 | Calcolo della somma di N numeri. |
| ⋮ ↓ | ⋮ ↓ | ⋮ ↓ | ⋮ ↓ |

I numeri d'ordine segnati nella prima colonna a sinistra stanno ad indicare le posizioni dei programmi sul nastro; nella seconda colonna sono riportati i nomi o le sigle dei programmi registrati, che **devono essere composti al massimo da 16 caratteri;** nella terza colonna è indicato il numero del contanastro segnalato sul registratore a cassetta al momento della registrazione di ogni programma; tale dato è molto utile per effettuare una ricerca veloce del programma stesso, utilizzando il tasto F.FWD; nell'ultima colonna, infine, è descritto in poche parole il contenuto di ciascun programma.

*2° ESERCIZIO*

Questo semplice esercizio di programmazione, utilizzante diverse istruzioni di diramazione, consente di individuare se un numero battuto sulla tastiera del computer è positivo oppure negativo o zero.

Possiamo affermare che le istruzioni di diramazione sono alla base di tutte le decisioni logiche che deve compiere il computer.

In particolare, la capacità di operare delle scelte è uno dei principali pregi che differenziano un computer da una comune calcolatrice.

Nella *fig. 3* è riportato il diagramma di flusso del programma in questione, che ora analizzeremo.

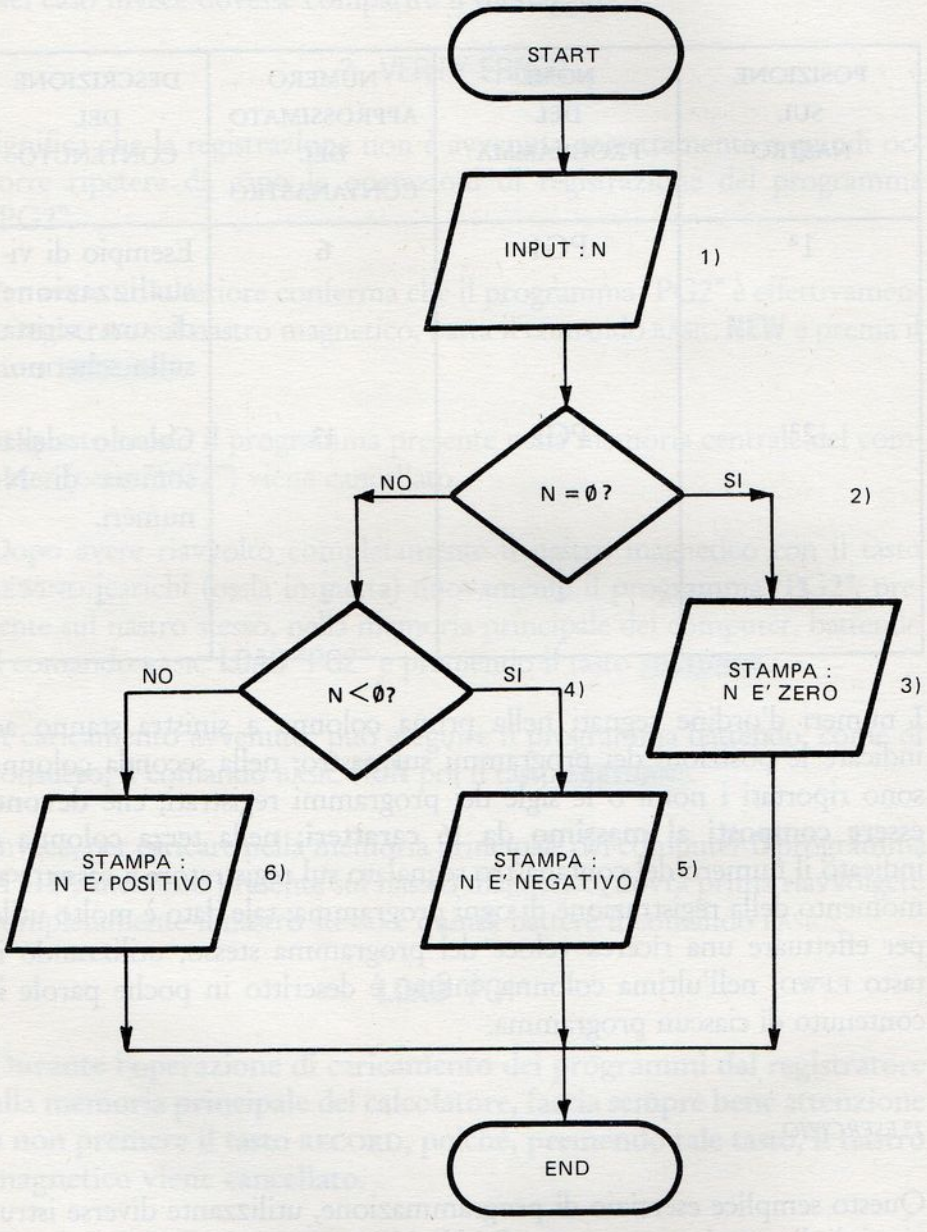



*Fig. 3 – Flow-chart del programma con istruzioni di diramazione.*

Il blocco 1) di tale diagramma viene fatto precedere da un'istruzione (*fig. 4*) che dice all'operatore di introdurre da tastiera un numero.

Ciò viene ottenuto scrivendo l'istruzione che segue:

**50 PRINT** "INTRODURRE UN NUMERO"

```
10 REM*******************************
20 REM* PROGRAMMA CON ISTRUZIONI *
30 REM*       DI DIRAMAZIONE       *
40 REM*******************************
50 PRINT"INTRODURRE UN NUMERO"
60 INPUT N
70 IF N=0 THEN 130
80 IF N<0 THEN 110
90 PRINT N;"E' POSITIVO"
100 GOTO 140
110 PRINT N;"E' NEGATIVO"
120 GOTO 140
130 PRINT N;"E' ZERO"
140 END
```

*Fig. 4 – Listato del programma con istruzioni di diramazione.*

Dopo questa istruzione di stampa segue l'istruzione di introduzione prevista dal blocco 1):

```
60 INPUT N
```

mediante la quale viene introdotto nel computer il numero.

Nel blocco 2) del diagramma della *fig. 3* l'elaborazione è posta di fronte ad un'alternativa semplice: proseguire per il blocco 3) se N=Ø, oppure proseguire per il blocco 4) se N non è uguale a zero (ossia se N è positivo o negativo).

L'istruzione che provvede a risolvere questa situazione è:

```
:
70 IF N=0 THEN 130
80
90
:
```

Se N=Ø, l'elaborazione prosegue per l'istruzione con numero di riferimento 13Ø, corrispondente al blocco 3), che visualizza (stampa su schermo) il numero Ø seguito dal messaggio E' ZERO ossia:

```
130 PRINT N;"E' ZERO"
```

Dopo di ciò il programma viene arrestato con l'apposita istruzione di

fine elaborazione:

```
140 END
```

Se invece N è diverso da zero, il programma prosegue per il blocco 4), dove si chiede se N<Ø (ossia se N è minore di Ø, cioè se è negativo).

L'istruzione che provvede ad eseguire questa seconda alternativa semplice è:

```
80 IF N<Ø THEN 110
```

Se N<Ø, l'elaborazione prosegue per l'istruzione 11Ø, relativa al blocco 5), che visualizza il numero N seguito dal messaggio: E' NEGATIVO.

Tale istruzione viene scritta nel modo che segue:

```
110 PRINT N; "E' NEGATIVO"
```

Dopo l'esecuzione del blocco 5), l'elaborazione termina.

Se invece non viene verificata la condizione N<Ø (ossia se N è maggiore di Ø, cioè è positivo), l'elaborazione prosegue per il blocco 6), senza che venga eseguito il blocco 5).

Il blocco 6) provvede a visualizzare il numero N seguito dal messaggio: E' POSITIVO.

Ciò viene ottenuto con la seguente istruzione:

```
90 PRINT N; "E' POSITIVO"
```

e quindi l'elaborazione si arresta.

Il listato del programma in questione è riportato nella *fig. 4*.

Possiamo ancora aggiungere a quanto è stato precisato che le istruzioni 1Ø÷4Ø sono di commento al programma e non influiscono sull'elaborazione.

Inoltre, come si può osservare, nel programma sono presenti due istruzioni di salto senza condizioni:

```
:
100 GOTO 140
:
120 GOTO 140
:
```

Queste due istruzioni provvedono a diramare l'elaborazione direttamente all'istruzione 14Ø, fine elaborazione, rispettivamente dopo le istruzioni di stampa sullo schermo 9Ø e 11Ø.

L'esame del programma è cosí terminato.

Mediante il comando **NEW** seguito da RETURN cancelli la memoria del computer e ripulisca lo schermo dalle scritte, premendo simultaneamente i tasti SHIFT e CLR | HOME .

Batta quindi sulla tastiera del computer il programma listato nella *fig. 4*.

Terminata la battitura, controlli attentamente il listato, confrontandolo con quello riportato nella *fig. 4*.

Se tutto è regolare, può passare ad eseguire il programma.

Batta pertanto il comando BASIC **RUN**, seguito da RETURN .

Vedrà apparire sullo schermo la scritta:

```
INTRODURRE UN NUMERO
?
```

Batta perciò sulla tastiera un numero qualsiasi, positivo o negativo, ad esempio –5, e prema il tasto RETURN .

Sullo schermo comparirà:

```
-5 E' NEGATIVO
```

Per introdurre altri numeri deve, per ognuno di essi, battere il comando **RUN** seguito da RETURN .

Se introdurrà un numero positivo, per esempio Ø.3, sullo schermo comparirà il messaggio:

```
Ø.3 E' POSITIVO
```

Se invece introdurrà il numero Ø, apparirà la scritta seguente:

Ø E' ZERO

*3° ESERCIZIO*

Il breve programma che segue consente di calcolare l'età che avrà una persona nell'anno 2ØØØ.

Il diagramma di flusso del programma in oggetto è riportato nella *fig. 5*.

Nel blocco 1) sono presenti due istruzioni di stampa, che provvedono a visualizzare sullo schermo i seguenti messaggi:

QUESTO PROGRAMMA CALCOLA L'ETA' CHE TU
AVRAI NEL 2ØØØ
DESIDERI CONOSCERLA

Nel blocco 2) è rappresentata un'istruzione **INPUT**, mediante la quale si può battere la parola SI se si vuole eseguire il calcolo, oppure battere la parola NO in caso contrario.

La risposta data, SI oppure NO, viene indicata nel programma con la variabile alfanumerica A$.

Nel blocco 3) il programma si chiede se A$=NO, ossia se con l'istruzione **INPUT** è stata battuta la parola NO. In caso affermativo, l'elaborazione termina.

Se la parola battuta non è NO, cioè è SI oppure qualsiasi altra parola, il programma prosegue per il blocco 4), dove è rappresentata una seconda **INPUT** che avverte l'operatore di introdurre il numero corrispondente all'anno di nascita.

Dopo di ciò nel blocco 5) viene eseguito il calcolo dell'età che si avrà nel 2ØØØ, la quale verrà visualizzata sullo schermo mediante il blocco 6).

Dopo la visualizzazione dell'età, l'elaborazione si arresta definitivamente.

Il listato del programma in questione è mostrato nella *fig. 6*.

Le istruzioni 1Ø e 2Ø eseguono la stampa dei due messaggi previsti nel

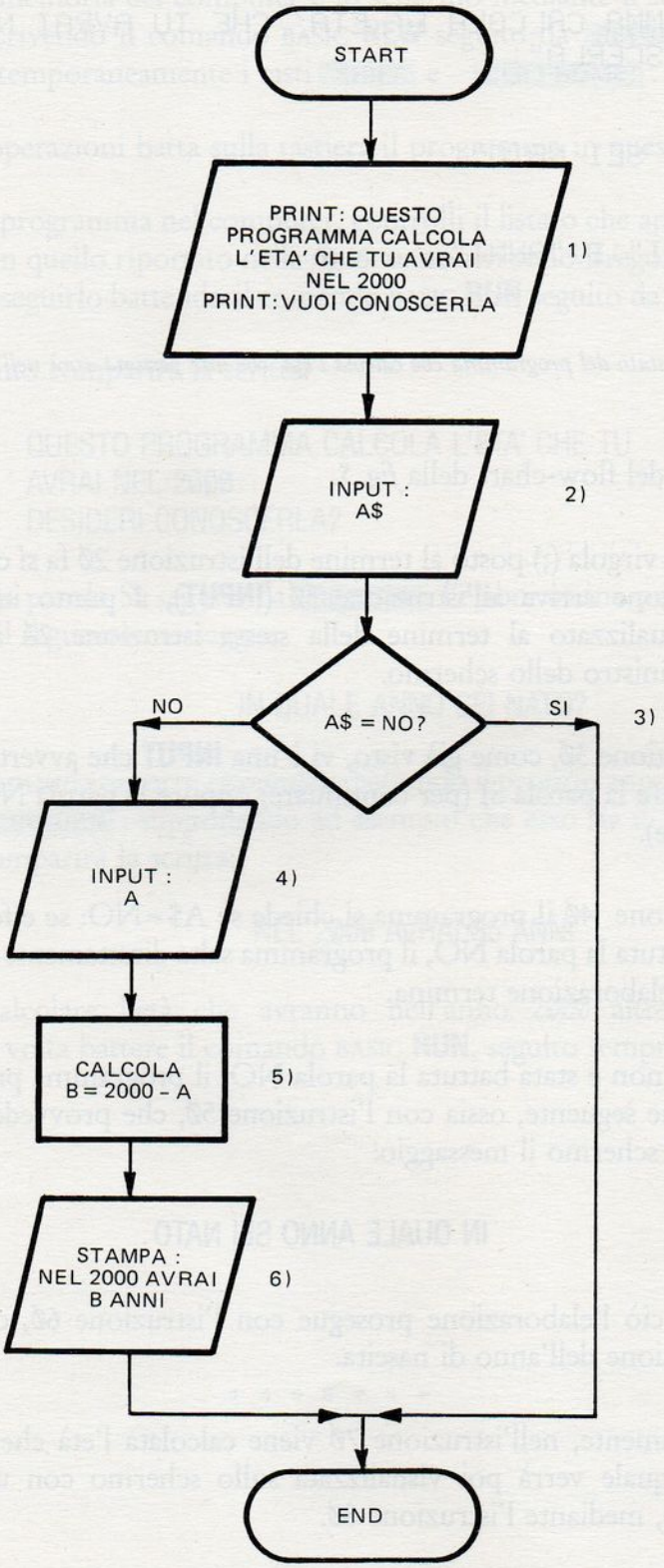



*Fig. 5 – Flow-chart del programma che calcola l'età di una persona nell'anno 2000.*

```
10 PRINT"QUESTO PROGRAMMA CALCOLA L' ETA' CHE TU AVRAI NEL 2000"
20 PRINT"DESIDERI CONOSCERLA";
30 INPUT A$
40 IF A$="NO" THEN 90
50 PRINT"IN QUALE ANNO SEI NATO";
60 INPUT A
70 B=2000-A
80 PRINT"NEL 2000 AVRAI";B;"ANNI"
90 END
```

*Fig. 6 – Listato del programma che calcola l'età che una persona avrà nell'anno 2000.*

blocco 1) del flow-chart della *fig. 5*.

Il punto e virgola (;) posto al termine dell'istruzione 20 fa sí che, quando l'elaborazione arriva all'istruzione 30 (**INPUT**), il punto interrogativo venga visualizzato al termine della stessa istruzione 20 anziché sul margine sinistro dello schermo.

Nell'istruzione 30, come già visto, vi è una **INPUT** che avverte l'operatore di battere la parola SI (per continuare) oppure la parola NO (per non continuare).

All'istruzione 40 il programma si chiede se A$=NO: se effettivamente è stata battuta la parola NO, il programma salta direttamente all'istruzione 90 e l'elaborazione termina.

Se invece non è stata battuta la parola NO, il programma prosegue con l'istruzione seguente, ossia con l'istruzione 50, che provvede a visualizzare sullo schermo il messaggio:

IN QUALE ANNO SEI NATO

Dopo di ciò l'elaborazione prosegue con l'istruzione 60, che prevede l'introduzione dell'anno di nascita.

Successivamente, nell'istruzione 70 viene calcolata l'età che si avrà nel 2000, la quale verrà poi visualizzata sullo schermo con un apposito messaggio, mediante l'istruzione 80.

L'analisi del programma è cosí conclusa.

Cancelli la memoria del computer e lo schermo mediante il solito sistema, ossia scrivendo il comando BASIC **NEW** seguito da RETURN e premendo contemporaneamente i tasti SHIFT e CLR | HOME .

Dopo tali operazioni batta sulla tastiera il programma in questione.

Immesso il programma nel computer, controlli il listato che appare sullo schermo con quello riportato nella *fig. 6*: se non vi sono irregolarità, può passare ad eseguirlo battendo il comando BASIC **RUN** seguito da RETURN .

Sullo schermo comparirà la scritta:

```
QUESTO PROGRAMMA CALCOLA L'ETA' CHE TU
AVRAI NEL 2000
DESIDERI CONOSCERLA?
```

Se batterà la parola SI seguita da RETURN , l'elaborazione proseguirà e comparirà il seguente messaggio:

```
IN QUALE ANNO SEI NATO?
```

A questa domanda occorre rispondere battendo il proprio anno di nascita seguito da RETURN ; supponendo ad esempio che esso sia il 1965, sullo schermo comparirà la scritta:

```
NEL 2000 AVRAI 35 ANNI
```

Volendo calcolare l'età che avranno nell'anno 2000 altre persone, dovrà ogni volta battere il comando BASIC **RUN**, seguito sempre dal tasto RETURN .

* * * * * * * *

## In seguito...

**Nella prossima lezione avrà modo di approfondire le Sue conoscenze sul linguaggio BASIC mediante una serie di esercitazioni sintattiche, riguardanti la correzione degli errori che piú comunemente si possono commettere durante l'esecuzione di un programma. Eseguirà inoltre esercizi di programmazione impieganti tutte le istruzioni finora studiate.**

# CONTROLLO DIDATTICO 3

**Domande**

1) - Quali sono le principali fasi della programmazione del computer?

.....

.....

.....

.....

.....

2) - Qual è il simbolo usato per raffigurare l'immissione di dati dalla tastiera e qual è quello per raffigurare l'emissione di un risultato su una stampante periferica?

3) - Con quale simbolo viene raffigurata un'alternativa semplice nel diagramma di flusso e con quale istruzione BASIC essa viene realizzata?

4) - Con quale istruzione BASIC viene realizzata l'immissione dei dati dalla tastiera?

.....

.....

5) - In che cosa differisce il linguaggio BASIC dagli altri linguaggi ad alto livello?

.....

.....

.....

6) - Da quale linguaggio evoluto deriva il linguaggio BASIC?

.....

7) - Che cosa si intende per "istruzione ad esecuzione immediata" e per "istruzione ad esecuzione differita"?

.....

.....

.....

.....

8) - Per "nome di una variabile" che cosa si intende e quali e quanti sono i caratteri che possono essere usati per rappresentare nomi di variabili?

.....

.....

.....

.....

.....

9) - Quali sono le variabili dotate di indici?

.....

.....

.....

.....

10) - Quale funzione svolgono i comandi BASIC?

.....

.....

.....

.....

11) - L'istruzione **INPUT** può essere ad esecuzione immediata?

.....

12) - Quali sono i comandi BASIC da Lei sinora usati?

.....

.....

.....

.....

.....

13) - Quale funzione svolge l'istruzione **REM**?

.....

.....

14) - Come sono memorizzate nella memoria principale del computer le matrici ed i vettori?

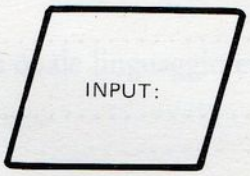
.....

.....

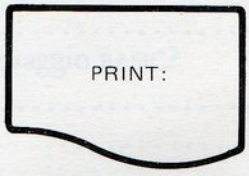
## Risposte

1) - Le fasi di programmazione sono essenzialmente tre: analisi del problema con stesura del flow-chart; codifica del programma in un linguaggio di programmazione; elaborazione vera e propria con emissione, da parte del computer, dei risultati richiesti.

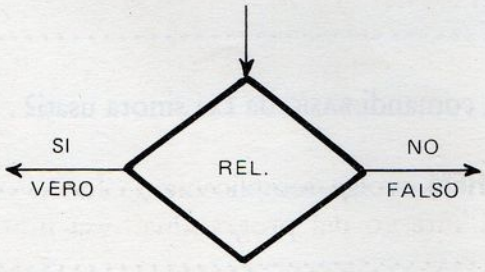
2) - Simbolo per l'immissione dei dati:




Simbolo per l'emissione dati su stampante:




3) - Simbolo dell'alternativa semplice di programma:



L'istruzione BASIC che realizza tale alternativa è: **IF...THEN...**

4) - L'immissione dei dati da tastiera si realizza con l'istruzione BASIC **INPUT** X, dove con la lettera X si indica il nome di un dato numerico generico.

5) - Il linguaggio BASIC si differenzia dagli altri linguaggi ad alto livello o evoluti perchè è di tipo "interpretato" e non "compilato".

6) - Il linguaggio BASIC deriva dal linguaggio FORTRAN.

7) - Un'istruzione ad esecuzione immediata è un enunciato BASIC non dotato di numero di riferimento; essa viene eseguita non appena si preme il tasto **RETURN** .
Un'istruzione od esecuzione differita è un enunciato BASIC dotato di numero di riferimento; viene eseguita durante il flusso di elaborazione del programma di cui fa parte.

8) - Per nome di una variabile si intende un raggruppamento di lettere e numeri (caratteri alfanumerici), che servono ad identificare il valore della variabile stessa.
Tale raggruppamento inizia sempre con una lettera dell'alfabeto e può essere lungo da due a sei caratteri alfanumerici.
Il nome di una variabile deve sempre terminare con il carattere $, se si tratta di una stringa, oppure con il carattere %, se si tratta di un numero intero.

9) - Le variabili dotate di indici sono i vettori e le matrici.

10) - I comandi BASIC hanno una funzione essenzialmente organizzativa; essi servono per effettuare operazioni di immissione ed emissione di informazioni da e verso i dispositivi periferici, per regolare il flusso di elaborazione, visualizzare archivi di dati e programmi, ecc.

11) - No. L'istruzione **INPUT** è ad esecuzione differita.

12) - I comandi BASIC sinora usati sono: **LIST, SAVE, RUN, LOAD, VERIFY, NEW.**

13) - L'istruzione **REM** svolge semplicemente funzioni descrittive e di commento all'interno del programma: essa non ha alcun valore operativo.

14) - Le matrici ed i vettori sono memorizzati in ordine sequenziale. Tale organizzazione consente al sistema un rapido reperimento di qualsiasi elemento.

* * * * * * *

# PARLA BASIC

Corso teorico-pratico per corrispondenza di

ANALISI E PROGRAMMAZIONE BASIC

**Scuola Radio Elettra**

10126 Torino - Via Stellone 5

Tel. (011) 674432